

Manual PostGIS 3.4.0beta1

Contents

1	Introducción	1
1.1	Comité de Dirección del Proyecto (Project Steering Committee)	1
1.2	Core Contributors Present	1
1.3	Core Contributors Past	2
1.4	Other Contributors	2
2	Instalación de PostGIS	5
2.1	Versión corta	5
2.2	Compilación e instalación desde el código fuente	5
2.2.1	Obteniendo el código fuente	6
2.2.2	Install Requirements	6
2.2.3	Configuración	7
2.2.4	Compilando	9
2.2.5	Compilando e Instalando Extensiones de PostGIS	9
2.2.6	Tests	11
2.2.7	Instalación	14
2.3	Installing and Using the address standardizer	15
2.4	Installing, Upgrading Tiger Geocoder, and loading data	15
2.4.1	Tiger Geocoder Enabling your PostGIS database	15
2.4.2	Using Address Standardizer Extension with Tiger geocoder	18
2.4.3	Required tools for tiger data loading	18
2.4.4	Upgrading your Tiger Geocoder Install and Data	18
2.5	Common Problems during installation	19
3	Administración de PostGIS	20
3.1	Performance Tuning	20
3.1.1	Startup	20
3.1.2	Runtime	21
3.2	Configuring raster support	21
3.3	Creating spatial databases	22

3.3.1	Spatially enable database using EXTENSION	22
3.3.2	Spatially enable database without using EXTENSION (discouraged)	22
3.4	Upgrading spatial databases	23
3.4.1	Soft upgrade	23
3.4.1.1	Soft Upgrade 9.1+ using extensions	23
3.4.1.2	Soft Upgrade Pre 9.1+ or without extensions	24
3.4.2	Hard upgrade	25
4	Data Management	27
4.1	Spatial Data Model	27
4.1.1	OGC Geometry	27
4.1.1.1	Point	28
4.1.1.2	LineString	28
4.1.1.3	LinearRing	28
4.1.1.4	Polygon	28
4.1.1.5	MultiPoint	28
4.1.1.6	MultiLineString	28
4.1.1.7	MultiPolygon	29
4.1.1.8	GeometryCollection	29
4.1.1.9	PolyhedralSurface	29
4.1.1.10	Triangle	29
4.1.1.11	TIN	29
4.1.2	SQL/MM Part 3 - Curves	29
4.1.2.1	CircularString	30
4.1.2.2	CompoundCurve	30
4.1.2.3	CurvePolygon	30
4.1.2.4	MultiCurve	30
4.1.2.5	MultiSurface	30
4.1.3	WKT and WKB	31
4.2	Geometry Data Type	32
4.2.1	PostGIS EWKB and EWKT	32
4.3	Geography Data Type	34
4.3.1	Creating Geography Tables	34
4.3.2	Using Geography Tables	35
4.3.3	When to use the Geography data type	36
4.3.4	Preguntas frecuentes Avanzadas de Geography	36
4.4	Geometry Validation	37
4.4.1	Simple Geometry	37
4.4.2	Valid Geometry	39

4.4.3	Managing Validity	41
4.5	Spatial Reference Systems	42
4.5.1	SPATIAL_REF_SYS Table	43
4.5.2	User-Defined Spatial Reference Systems	44
4.6	Spatial Tables	44
4.6.1	Crear una tabla espacial	44
4.6.2	GEOMETRY_COLUMNS View	45
4.6.3	Manually Registering Geometry Columns	46
4.7	Loading Spatial Data	48
4.7.1	Using SQL to Load Data	48
4.7.2	Using the Shapefile Loader	48
4.8	Extracting Spatial Data	50
4.8.1	Using SQL to Extract Data	50
4.8.2	Using the Shapefile Dumper	51
4.9	Spatial Indexes	51
4.9.1	Indices GiST	52
4.9.2	BRIN Indexes	52
4.9.3	SP-GiST Indexes	54
4.9.4	Tuning Index Usage	55
5	Consulta Espacial	56
5.1	Determining Spatial Relationships	56
5.1.1	Dimensionally Extended 9-Intersection Model	56
5.1.2	Named Spatial Relationships	58
5.1.3	General Spatial Relationships	59
5.2	Using Spatial Indexes	61
5.3	Examples of Spatial SQL	61
6	Consejos de rendimiento	64
6.1	Tablas pequeñas de geometrías grandes	64
6.1.1	Descripción del problema	64
6.1.2	Soluciones provisionales	64
6.2	CLUSTERing o índices geométricos	65
6.3	Evitar la conversión de dimensión	65

7	Manual de Referencia PostGIS	66
7.1	Tipos Geometry/Geography/Box en PostgreSQL PostGIS	66
7.1.1	box2d	66
7.1.2	box3d	67
7.1.3	geometry	67
7.1.4	geometry_dump	68
7.1.5	geography	68
7.2	Funciones de Gestión	68
7.2.1	AddGeometryColumn	68
7.2.2	DropGeometryColumn	70
7.2.3	DropGeometryTable	71
7.2.4	Find_SRID	72
7.2.5	Populate_Geometry_Columns	72
7.2.6	UpdateGeometrySRID	74
7.3	Constructores Geométricos	75
7.3.1	ST_GeomCollFromText	75
7.3.2	ST_LineFromMultiPoint	77
7.3.3	ST_MakeEnvelope	77
7.3.4	ST_MakeLine	78
7.3.5	ST_MakePoint	80
7.3.6	ST_MakePointM	81
7.3.7	ST_MakePolygon	82
7.3.8	ST_Point	83
7.3.9	ST_Point	85
7.3.10	ST_Point	85
7.3.11	ST_Point	86
7.3.12	ST_Polygon	86
7.3.13	ST_MakeEnvelope	87
7.3.14	ST_HexagonGrid	88
7.3.15	ST_Hexagon	91
7.3.16	ST_SquareGrid	92
7.3.17	ST_Square	93
7.3.18	ST_Letters	94
7.4	Métodos de Acceso a Geometrías	95
7.4.1	GeometryType	95
7.4.2	ST_Boundary	96
7.4.3	ST_BoundingDiagonal	98
7.4.4	ST_CoordDim	99
7.4.5	ST_Dimension	100

7.4.6	ST_Dump	100
7.4.7	ST_NumPoints	102
7.4.8	ST_NumPoints	106
7.4.9	ST_NRings	108
7.4.10	ST_EndPoint	109
7.4.11	ST_Envelope	110
7.4.12	ST_ExteriorRing	112
7.4.13	ST_GeometryN	113
7.4.14	ST_GeometryType	115
7.4.15	ST_HasArc	116
7.4.16	ST_InteriorRingN	117
7.4.17	ST_IsClosed	118
7.4.18	ST_IsCollection	119
7.4.19	ST_IsEmpty	120
7.4.20	ST_IsPolygonCCW	122
7.4.21	ST_IsPolygonCW	122
7.4.22	ST_IsRing	123
7.4.23	ST_IsSimple	124
7.4.24	ST_M	125
7.4.25	ST_MemSize	125
7.4.26	ST_NDims	127
7.4.27	ST_NPoints	127
7.4.28	ST_NRings	128
7.4.29	ST_NumGeometries	128
7.4.30	ST_NumInteriorRings	129
7.4.31	ST_NumInteriorRing	130
7.4.32	ST_NumPatches	130
7.4.33	ST_NumPoints	131
7.4.34	ST_PatchN	131
7.4.35	ST_PointN	132
7.4.36	ST_Points	134
7.4.37	ST_StartPoint	135
7.4.38	ST_Summary	136
7.4.39	ST_X	137
7.4.40	ST_Y	138
7.4.41	ST_Z	138
7.4.42	ST_Zmflag	139
7.5	Editores de Geometría	140
7.5.1	ST_AddPoint	140

7.5.2	ST_CollectionExtract	141
7.5.3	ST_CollectionHomogenize	142
7.5.4	ST_CurveToLine	143
7.5.5	ST_Scroll	146
7.5.6	ST_FlipCoordinates	147
7.5.7	ST_Force2D	147
7.5.8	ST_Force3D	148
7.5.9	ST_Force3DZ	149
7.5.10	ST_Force3DM	150
7.5.11	ST_Force4D	150
7.5.12	ST_ForcePolygonCCW	151
7.5.13	ST_ForceCollection	152
7.5.14	ST_ForcePolygonCW	153
7.5.15	ST_ForceSFS	153
7.5.16	ST_ForceRHR	154
7.5.17	ST_ForceCurve	155
7.5.18	ST_LineToCurve	155
7.5.19	ST_Multi	157
7.5.20	ST_LineExtend	157
7.5.21	ST_Normalize	158
7.5.22	ST_Project	158
7.5.23	ST_QuantizeCoordinates	159
7.5.24	ST_RemovePoint	161
7.5.25	ST_RemoveRepeatedPoints	162
7.5.26	ST_Reverse	163
7.5.27	ST_Segmentize	163
7.5.28	ST_SetPoint	165
7.5.29	ST_ShiftLongitude	166
7.5.30	ST_WrapX	167
7.5.31	ST_SnapToGrid	168
7.5.32	ST_Snap	169
7.5.33	ST_SwapOrdinates	172
7.6	Geometry Validation	173
7.6.1	ST_IsValid	173
7.6.2	ST_IsValidDetail	174
7.6.3	ST_IsValidReason	176
7.6.4	ST_MakeValid	177
7.7	Spatial Reference System Functions	182
7.7.1	ST_InverseTransformPipeline	182

7.7.2	ST_SetSRID	183
7.7.3	ST_SRID	184
7.7.4	ST_Transform	185
7.7.5	ST_TransformPipeline	187
7.7.6	postgis_srs_codes	189
7.7.7	postgis_srs	189
7.7.8	postgis_srs_all	190
7.7.9	postgis_srs_search	191
7.8	Geometry Input	192
7.8.1	Well-Known Text (WKT)	192
7.8.1.1	ST_BdPolyFromText	192
7.8.1.2	ST_BdMPolyFromText	192
7.8.1.3	ST_GeogFromText	193
7.8.1.4	ST_GeographyFromText	193
7.8.1.5	ST_GeomCollFromText	194
7.8.1.6	ST_GeomFromEWKT	194
7.8.1.7	ST_GeomFromMARC21	196
7.8.1.8	ST_GeometryFromText	198
7.8.1.9	ST_GeomFromText	199
7.8.1.10	ST_LineFromText	200
7.8.1.11	ST_MLineFromText	201
7.8.1.12	ST_MPointFromText	202
7.8.1.13	ST_MPolyFromText	202
7.8.1.14	ST_PointFromText	203
7.8.1.15	ST_PolygonFromText	204
7.8.1.16	ST_WKTTToSQL	205
7.8.2	Well-Known Binary (WKB)	205
7.8.2.1	ST_GeogFromWKB	205
7.8.2.2	ST_GeomFromEWKB	206
7.8.2.3	ST_GeomFromWKB	207
7.8.2.4	ST_LineFromWKB	208
7.8.2.5	ST_LinestringFromWKB	209
7.8.2.6	ST_PointFromWKB	210
7.8.2.7	ST_WKBTToSQL	211
7.8.3	Other Formats	211
7.8.3.1	ST_Box2dFromGeoHash	211
7.8.3.2	ST_GeomFromGeoHash	212
7.8.3.3	ST_GeomFromGML	213
7.8.3.4	ST_GeomFromGeoJSON	215

7.8.3.5	ST_GeomFromKML	216
7.8.3.6	ST_GeomFromTWKB	217
7.8.3.7	ST_GMLToSQL	218
7.8.3.8	ST_LineFromEncodedPolyline	218
7.8.3.9	ST_PointFromGeoHash	219
7.8.3.10	ST_FromFlatGeobufToTable	220
7.8.3.11	ST_FromFlatGeobuf	220
7.9	Geometry Output	220
7.9.1	Well-Known Text (WKT)	220
7.9.1.1	ST_AsEWKT	220
7.9.1.2	ST_AsText	222
7.9.2	Well-Known Binary (WKB)	223
7.9.2.1	ST_AsBinary	223
7.9.2.2	ST_AsEWKB	224
7.9.2.3	ST_AsHEXEWKB	225
7.9.3	Other Formats	226
7.9.3.1	ST_AsEncodedPolyline	226
7.9.3.2	ST_AsFlatGeobuf	227
7.9.3.3	ST_AsGeobuf	228
7.9.3.4	ST_AsGeoJSON	228
7.9.3.5	ST_AsGML	230
7.9.3.6	ST_AsKML	234
7.9.3.7	ST_AsLatLonText	235
7.9.3.8	ST_AsMARC21	236
7.9.3.9	ST_AsMVTGeom	239
7.9.3.10	ST_AsMVT	240
7.9.3.11	ST_AsSVG	241
7.9.3.12	ST_AsTWKB	242
7.9.3.13	ST_AsX3D	243
7.9.3.14	ST_GeoHash	247
7.10	Operadores	248
7.10.1	Bounding Box Operators	248
7.10.1.1	&&	248
7.10.1.2	&&(geometry,box2df)	249
7.10.1.3	&&(box2df,geometry)	250
7.10.1.4	&&(box2df,box2df)	250
7.10.1.5	&&&	251
7.10.1.6	&&&(geometry,gidx)	252
7.10.1.7	&&&(gidx,geometry)	253

7.10.1.8	&&&(gidx,gidx)	254
7.10.1.9	&<	255
7.10.1.10	&<	256
7.10.1.11	&>	256
7.10.1.12	<<	257
7.10.1.13	<<	258
7.10.1.14	=	259
7.10.1.15	>>	260
7.10.1.16	@	261
7.10.1.17	@(geometry,box2df)	261
7.10.1.18	@(box2df,geometry)	262
7.10.1.19	@(box2df,box2df)	263
7.10.1.20	&>	264
7.10.1.21	>>	264
7.10.1.22	~	265
7.10.1.23	~(geometry,box2df)	266
7.10.1.24	~(box2df,geometry)	267
7.10.1.25	~(box2df,box2df)	267
7.10.1.26	~=	268
7.10.2	Operadores	269
7.10.2.1	<->	269
7.10.2.2	=	271
7.10.2.3	<#>	272
7.10.2.4	<<->>	273
7.10.2.5	<<#>>	273
7.11	Spatial Relationships	274
7.11.1	Topological Relationships	274
7.11.1.1	ST_3DIntersects	274
7.11.1.2	ST_Contains	275
7.11.1.3	ST_ContainsProperly	279
7.11.1.4	ST_CoveredBy	280
7.11.1.5	ST_Covers	281
7.11.1.6	ST_Crosses	283
7.11.1.7	ST_Disjoint	285
7.11.1.8	ST_Equals	286
7.11.1.9	ST_Intersects	287
7.11.1.10	ST_LineCrossingDirection	289
7.11.1.11	ST_OrderingEquals	291
7.11.1.12	ST_Overlaps	292

7.11.1.13	ST_Relate	295
7.11.1.14	ST_RelateMatch	298
7.11.1.15	ST_Touches	299
7.11.1.16	ST_Within	300
7.11.2	Distance Relationships	302
7.11.2.1	ST_3DDWithin	302
7.11.2.2	ST_3DDFullyWithin	303
7.11.2.3	ST_DFullyWithin	304
7.11.2.4	ST_DWithin	305
7.11.2.5	ST_PointInsideCircle	306
7.12	Measurement Functions	307
7.12.1	ST_Area	307
7.12.2	ST_Azimuth	308
7.12.3	ST_Angle	310
7.12.4	ST_ClosestPoint	311
7.12.5	ST_3DClosestPoint	312
7.12.6	ST_Distance	313
7.12.7	ST_3DDistance	315
7.12.8	ST_DistanceSphere	316
7.12.9	ST_DistanceSpheroid	317
7.12.10	ST_FrechetDistance	318
7.12.11	ST_HausdorffDistance	319
7.12.12	ST_Length	320
7.12.13	ST_Length2D	322
7.12.14	ST_3DLength	322
7.12.15	ST_LengthSpheroid	323
7.12.16	ST_LongestLine	324
7.12.17	ST_3DLongestLine	326
7.12.18	ST_MaxDistance	327
7.12.19	ST_3DMaxDistance	328
7.12.20	ST_MinimumClearance	329
7.12.21	ST_MinimumClearanceLine	330
7.12.22	ST_Perimeter	330
7.12.23	ST_Perimeter2D	332
7.12.24	ST_3DPerimeter	332
7.12.25	ST_ShortestLine	333
7.12.26	ST_3DShortestLine	335
7.13	Overlay Functions	336
7.13.1	ST_ClipByBox2D	336

7.13.2	ST_Difference	337
7.13.3	ST_Intersection	338
7.13.4	ST_MemUnion	341
7.13.5	ST_Node	341
7.13.6	ST_Split	342
7.13.7	ST_Subdivide	345
7.13.8	ST_SymDifference	347
7.13.9	ST_UnaryUnion	348
7.13.10	ST_Union	349
7.14	Procesamiento de geometría	351
7.14.1	ST_Buffer	351
7.14.2	ST_BuildArea	356
7.14.3	ST_Centroid	357
7.14.4	ST_ChaikinSmoothing	359
7.14.5	ST_ConcaveHull	361
7.14.6	ST_ConvexHull	364
7.14.7	ST_DelaunayTriangles	366
7.14.8	ST_FilterByM	371
7.14.9	ST_GeneratePoints	372
7.14.10	ST_GeometricMedian	372
7.14.11	ST_LineMerge	374
7.14.12	ST_MaximumInscribedCircle	376
7.14.13	ST_LargestEmptyCircle	378
7.14.14	ST_MinimumBoundingCircle	380
7.14.15	ST_MinimumBoundingRadius	381
7.14.16	ST_OrientedEnvelope	382
7.14.17	ST_OffsetCurve	383
7.14.18	ST_PointOnSurface	387
7.14.19	ST_Polygonize	389
7.14.20	ST_ReducePrecision	390
7.14.21	ST_SharedPaths	392
7.14.22	ST_Simplify	394
7.14.23	ST_SimplifyPreserveTopology	395
7.14.24	ST_SimplifyPolygonHull	395
7.14.25	ST_SimplifyVW	398
7.14.26	ST_SetEffectiveArea	398
7.14.27	ST_TriangulatePolygon	400
7.14.28	ST_VoronoiLines	401
7.14.29	ST_VoronoiPolygons	402

7.15	Coverages	404
7.15.1	ST_CoverageInvalidEdges	404
7.15.2	ST_CoverageSimplify	406
7.15.3	ST_CoverageUnion	407
7.16	Affine Transformations	408
7.16.1	ST_Affine	408
7.16.2	ST_Rotate	410
7.16.3	ST_RotateX	411
7.16.4	ST_RotateY	412
7.16.5	ST_RotateZ	413
7.16.6	ST_Scale	414
7.16.7	ST_Translate	415
7.16.8	ST_TransScale	416
7.17	Clustering Functions	417
7.17.1	ST_ClusterDBSCAN	417
7.17.2	ST_ClusterIntersectingWin	420
7.17.3	ST_ClusterIntersecting	420
7.17.4	ST_ClusterKMeans	421
7.17.5	ST_ClusterWithin	423
7.17.6	ST_ClusterWithinWin	424
7.18	Bounding Box Functions	425
7.18.1	Box2D	425
7.18.2	Box3D	426
7.18.3	ST_EstimatedExtent	426
7.18.4	ST_Expand	427
7.18.5	ST_Extent	429
7.18.6	ST_3DExtent	430
7.18.7	ST_MakeBox2D	431
7.18.8	ST_3DMakeBox	432
7.18.9	ST_XMax	432
7.18.10	ST_XMin	433
7.18.11	ST_YMax	434
7.18.12	ST_YMin	435
7.18.13	ST_ZMax	436
7.18.14	ST_ZMin	437
7.19	Referencia Lineal	438
7.19.1	ST_LineInterpolatePoint	438
7.19.2	ST_LineInterpolatePoint	440
7.19.3	ST_LineInterpolatePoints	440

7.19.4	ST_LineLocatePoint	441
7.19.5	ST_LineSubstring	442
7.19.6	ST_LocateAlong	444
7.19.7	ST_LocateBetween	445
7.19.8	ST_LocateBetweenElevations	446
7.19.9	ST_InterpolatePoint	447
7.19.10	ST_AddMeasure	448
7.20	Trajectory Functions	449
7.20.1	ST_IsValidTrajectory	449
7.20.2	ST_ClosestPointOfApproach	449
7.20.3	ST_DistanceCPA	450
7.20.4	ST_CPAWithin	451
7.21	SFCGAL Functions	452
7.21.1	postgis_sfcgal_version	452
7.21.2	postgis_sfcgal_full_version	452
7.21.3	ST_3DArea	453
7.21.4	ST_3DConvexHull	454
7.21.5	ST_3DIntersection	455
7.21.6	ST_3DDifference	457
7.21.7	ST_3DUnion	458
7.21.8	ST_AlphaShape	459
7.21.9	ST_ApproximateMedialAxis	462
7.21.10	ST_ConstrainedDelaunayTriangles	463
7.21.11	ST_Extrude	464
7.21.12	ST_ForceLHR	466
7.21.13	ST_IsPlanar	466
7.21.14	ST_IsSolid	466
7.21.15	ST_MakeSolid	467
7.21.16	ST_MinkowskiSum	467
7.21.17	ST_OptimalAlphaShape	469
7.21.18	ST_Orientation	471
7.21.19	ST_StraightSkeleton	472
7.21.20	ST_Tessellate	473
7.21.21	ST_Volume	475
7.22	Soporte para transacciones grandes	476
7.22.1	AddAuth	476
7.22.2	CheckAuth	477
7.22.3	DisableLongTransactions	478
7.22.4	EnableLongTransactions	478

7.22.5	LockRow	479
7.22.6	UnlockRows	479
7.23	Version Functions	480
7.23.1	PostGIS_Extensions_Upgrade	480
7.23.2	PostGIS_Full_Version	481
7.23.3	PostGIS_GEOS_Version	481
7.23.4	PostGIS_GEOS_Compiled_Version	482
7.23.5	PostGIS_Liblwgeom_Version	482
7.23.6	PostGIS_LibXML_Version	483
7.23.7	PostGIS_Lib_Build_Date	483
7.23.8	PostGIS_Lib_Version	484
7.23.9	PostGIS_PROJ_Version	484
7.23.10	PostGIS_Wagyu_Version	485
7.23.11	PostGIS_Scripts_Build_Date	485
7.23.12	PostGIS_Scripts_Installed	486
7.23.13	PostGIS_Scripts_Released	486
7.23.14	PostGIS_Version	487
7.24	Grand Unified Custom Variables (GUCs)	488
7.24.1	postgis.backend	488
7.24.2	postgis.gdal_datapath	488
7.24.3	postgis.gdal_enabled_drivers	489
7.24.4	postgis.enable_outdb_rasters	490
7.24.5	postgis.gdal_config_options	491
7.25	Troubleshooting Functions	492
7.25.1	PostGIS_AddBBBox	492
7.25.2	PostGIS_DropBBBox	492
7.25.3	PostGIS_HasBBBox	493
8	Topology	495
8.1	Tipos en Topology	495
8.1.1	getfaceedges_returntype	495
8.1.2	TopoGeometry	496
8.1.3	validatetopology_returntype	496
8.2	Dominios de Topology	497
8.2.1	TopoElement	497
8.2.2	TopoElementArray	497
8.3	Topología y Gestión de TopoGeometría	498
8.3.1	AddTopoGeometryColumn	498
8.3.2	RenameTopoGeometryColumn	499

8.3.3	DropTopology	499
8.3.4	RenameTopology	500
8.3.5	DropTopoGeometryColumn	500
8.3.6	Populate_Topology_Layer	501
8.3.7	TopologySummary	502
8.3.8	ValidateTopology	503
8.3.9	ValidateTopologyRelation	505
8.3.10	FindTopology	505
8.3.11	FindLayer	506
8.4	Topology Statistics Management	506
8.5	Constructores de Topología	506
8.5.1	CreateTopology	506
8.5.2	CopyTopology	507
8.5.3	ST_InitTopoGeo	508
8.5.4	ST_CreateTopoGeo	509
8.5.5	TopoGeo_AddPoint	509
8.5.6	TopoGeo_AddLineString	510
8.5.7	TopoGeo_AddPolygon	510
8.6	Editores de Topología	511
8.6.1	ST_AddIsoNode	511
8.6.2	ST_AddIsoEdge	511
8.6.3	ST_AddEdgeNewFaces	512
8.6.4	ST_AddEdgeModFace	513
8.6.5	ST_RemEdgeNewFace	513
8.6.6	ST_RemEdgeModFace	514
8.6.7	ST_ChangeEdgeGeom	515
8.6.8	ST_ModEdgeSplit	515
8.6.9	ST_ModEdgeHeal	516
8.6.10	ST_NewEdgeHeal	517
8.6.11	ST_MoveIsoNode	517
8.6.12	ST_NewEdgesSplit	518
8.6.13	ST_RemoveIsoNode	519
8.6.14	ST_RemoveIsoEdge	519
8.7	Accesores de Topología	520
8.7.1	GetEdgeByPoint	520
8.7.2	GetFaceByPoint	521
8.7.3	GetFaceContainingPoint	521
8.7.4	GetNodeByPoint	522
8.7.5	GetTopologyID	523

8.7.6	GetTopologySRID	523
8.7.7	GetTopologyName	524
8.7.8	ST_GetFaceEdges	524
8.7.9	ST_GetFaceGeometry	525
8.7.10	GetRingEdges	526
8.7.11	GetNodeEdges	526
8.8	Procesamiento de Topología	527
8.8.1	Polygonize	527
8.8.2	AddNode	528
8.8.3	AddEdge	528
8.8.4	AddFace	529
8.8.5	ST_Simplify	531
8.8.6	RemoveUnusedPrimitives	532
8.9	Constructores de Geometría Topográfica	532
8.9.1	CreateTopoGeom	532
8.9.2	toTopoGeom	534
8.9.3	TopoElementArray_Agg	535
8.9.4	TopoElement	535
8.10	Editores TopoGeometry	536
8.10.1	clearTopoGeom	536
8.10.2	TopoGeom_addElement	537
8.10.3	TopoGeom_remElement	537
8.10.4	TopoGeom_addTopoGeom	537
8.10.5	toTopoGeom	538
8.11	Descriptores de Geometría Topográfica	538
8.11.1	GetTopoGeomElementArray	538
8.11.2	GetTopoGeomElements	539
8.11.3	ST_SRID	539
8.12	Salidas de Geometría Topográfica	540
8.12.1	AsGML	540
8.12.2	AsTopoJSON	542
8.13	Relaciones espaciales de topología	544
8.13.1	Equals	544
8.13.2	Intersects	545
8.14	Importing and exporting Topologies	545
8.14.1	Using the Topology exporter	546
8.14.2	Using the Topology importer	546

9	Gestión, Consulta y Aplicaciones de Datos Raster	547
9.1	Cargando y Creando Rasters	547
9.1.1	Utilizar el paquete raster2pgsql para cargar rasters	547
9.1.1.1	Example Usage	547
9.1.1.2	raster2pgsql options	548
9.1.2	Crear rastres utilizando las funciones raster de PostGIS	549
9.1.3	Using "out db" cloud rasters	550
9.2	Catalogos raster	551
9.2.1	Catalogo de columnas raster	551
9.2.2	Previsualizaciones raster	552
9.3	Contruyendo aplicaciones personalizadas con PostGIS Raster	553
9.3.1	Ejemplo de salida utilizando ST_AsPNG junto con otras opciones raster en PHP	553
9.3.2	Ejemplo de salida utilizando ST_AsPNG junto con otras opciones raster en ASP.NET C#	554
9.3.3	Aplicación de consola Java que extrae un raster como un fichero de imagen	555
9.3.4	Utilizar PLPython para extraer imágenes vía SQL	557
9.3.5	Extraer un raster con PSQL	557
10	Raster Reference	558
10.1	Raster Support Data types	559
10.1.1	geomval	559
10.1.2	addbandarg	559
10.1.3	rastbandarg	559
10.1.4	raster	560
10.1.5	reclassarg	560
10.1.6	summarystats	561
10.1.7	unionarg	561
10.2	Raster Management	562
10.2.1	AddRasterConstraints	562
10.2.2	DropRasterConstraints	563
10.2.3	AddOverviewConstraints	564
10.2.4	DropOverviewConstraints	565
10.2.5	PostGIS_GDAL_Version	566
10.2.6	PostGIS_Raster_Lib_Build_Date	566
10.2.7	PostGIS_Raster_Lib_Version	567
10.2.8	ST_GDALDrivers	567
10.2.9	ST_Contour	572
10.2.10	ST_InterpolateRaster	573
10.2.11	UpdateRasterSRID	573
10.2.12	ST_CreateOverview	574

10.3 Raster Constructors	575
10.3.1 ST_AddBand	575
10.3.2 ST_AsRaster	577
10.3.3 ST_Band	579
10.3.4 ST_MakeEmptyCoverage	581
10.3.5 ST_MakeEmptyRaster	582
10.3.6 ST_Tile	583
10.3.7 ST_Retile	585
10.3.8 ST_FromGDALRaster	585
10.4 Raster Accessors	586
10.4.1 ST_GeoReference	586
10.4.2 ST_Height	587
10.4.3 ST_IsEmpty	588
10.4.4 ST_MemSize	588
10.4.5 ST_MetaData	589
10.4.6 ST_NumBands	590
10.4.7 ST_PixelHeight	590
10.4.8 ST_PixelWidth	591
10.4.9 ST_ScaleX	592
10.4.10 ST_ScaleY	593
10.4.11 ST_RasterToWorldCoord	594
10.4.12 ST_RasterToWorldCoordX	594
10.4.13 ST_RasterToWorldCoordY	595
10.4.14 ST_Rotation	596
10.4.15 ST_SkewX	597
10.4.16 ST_SkewY	598
10.4.17 ST_SRID	598
10.4.18 ST_Summary	599
10.4.19 ST_UpperLeftX	600
10.4.20 ST_UpperLeftY	600
10.4.21 ST_Width	601
10.4.22 ST_WorldToRasterCoord	601
10.4.23 ST_WorldToRasterCoordX	602
10.4.24 ST_WorldToRasterCoordY	603
10.5 Raster Band Accessors	603
10.5.1 ST_BandMetaData	603
10.5.2 ST_BandNoDataValue	605
10.5.3 ST_BandIsNoData	605
10.5.4 ST_BandPath	607

10.5.5	ST_BandFileSize	607
10.5.6	ST_BandFileTimestamp	608
10.5.7	ST_BandPixelType	608
10.5.8	ST_MinPossibleValue	609
10.5.9	ST_HasNoBand	610
10.6	Raster Pixel Accessors and Setters	610
10.6.1	ST_PixelAsPolygon	610
10.6.2	ST_PixelAsPolygons	611
10.6.3	ST_PixelAsPoint	612
10.6.4	ST_PixelAsPoints	613
10.6.5	ST_PixelAsCentroid	614
10.6.6	ST_PixelAsCentroids	614
10.6.7	ST_Value	615
10.6.8	ST_NearestValue	618
10.6.9	ST_SetZ	620
10.6.10	ST_SetM	621
10.6.11	ST_Neighborhood	623
10.6.12	ST_SetValue	625
10.6.13	ST_SetValues	626
10.6.14	ST_DumpValues	634
10.6.15	ST_PixelOfValue	635
10.7	Raster Editors	636
10.7.1	ST_SetGeoReference	636
10.7.2	ST_SetRotation	637
10.7.3	ST_SetScale	638
10.7.4	ST_SetSkew	639
10.7.5	ST_SetSRID	640
10.7.6	ST_SetUpperLeft	640
10.7.7	ST_Resample	641
10.7.8	ST_Rescale	642
10.7.9	ST_Reskew	644
10.7.10	ST_SnapToGrid	645
10.7.11	ST_Resize	646
10.7.12	ST_Transform	647
10.8	Raster Band Editors	650
10.8.1	ST_SetBandNoDataValue	650
10.8.2	ST_SetBandIsNoData	651
10.8.3	ST_SetBandPath	652
10.8.4	ST_SetBandIndex	654

10.9 Raster Band Statistics and Analytics	655
10.9.1 ST_Count	655
10.9.2 ST_CountAgg	656
10.9.3 ST_Histogram	657
10.9.4 ST_Quantile	659
10.9.5 ST_SummaryStats	661
10.9.6 ST_SummaryStatsAgg	662
10.9.7 ST_ValueCount	664
10.10 Raster Inputs	666
10.10.1 ST_RastFromWKB	666
10.10.2 ST_RastFromHexWKB	667
10.11 Raster Outputs	668
10.11.1 ST_AsBinary/ST_AsWKB	668
10.11.2 ST_AsHexWKB	668
10.11.3 ST_AsGDALRaster	669
10.11.4 ST_AsJPEG	670
10.11.5 ST_AsPNG	671
10.11.6 ST_AsTIFF	672
10.12 Raster Processing: Map Algebra	673
10.12.1 ST_Clip	673
10.12.2 ST_ColorMap	676
10.12.3 ST_Grayscale	679
10.12.4 ST_Intersection	681
10.12.5 ST_MapAlgebra (callback function version)	682
10.12.6 ST_MapAlgebra (expression version)	689
10.12.7 ST_MapAlgebraExpr	691
10.12.8 ST_MapAlgebraExpr	693
10.12.9 ST_MapAlgebraFct	698
10.12.10 ST_MapAlgebraFct	702
10.12.11 ST_MapAlgebraFctNgb	706
10.12.12 ST_Reclass	708
10.12.13 ST_Union	709
10.13 Built-in Map Algebra Callback Functions	711
10.13.1 ST_Distinct4ma	711
10.13.2 ST_InvDistWeight4ma	712
10.13.3 ST_Max4ma	712
10.13.4 ST_Mean4ma	713
10.13.5 ST_Min4ma	715
10.13.6 ST_MinDist4ma	716

10.13.7 ST_Range4ma	716
10.13.8 ST_StdDev4ma	717
10.13.9 ST_Sum4ma	718
10.14 Raster Processing: DEM (Elevation)	719
10.14.1 ST_Aspect	719
10.14.2 ST_HillShade	721
10.14.3 ST_Roughness	723
10.14.4 ST_Slope	723
10.14.5 ST_TPI	725
10.14.6 ST_TRI	726
10.15 Raster Processing: Raster to Geometry	726
10.15.1 Box3D	726
10.15.2 ST_ConvexHull	727
10.15.3 ST_DumpAsPolygons	728
10.15.4 ST_Envelope	729
10.15.5 ST_MinConvexHull	730
10.15.6 ST_Polygon	731
10.16 Raster Operators	732
10.16.1 &&	732
10.16.2 &<	733
10.16.3 &>	733
10.16.4 = ...	734
10.16.5 @	735
10.16.6 ~=	735
10.16.7 ~	736
10.17 Raster and Raster Band Spatial Relationships	736
10.17.1 ST_Contains	736
10.17.2 ST_ContainsProperly	737
10.17.3 ST_Covers	738
10.17.4 ST_CoveredBy	739
10.17.5 ST_Disjoint	740
10.17.6 ST_Intersects	741
10.17.7 ST_Overlaps	742
10.17.8 ST_Touches	743
10.17.9 ST_SameAlignment	743
10.17.10 ST_NotSameAlignmentReason	744
10.17.11 ST_Within	745
10.17.12 ST_DWithin	746
10.17.13 ST_DFullyWithin	747

10.18 Raster Tips	748
10.18.1 Out-DB Rasters	748
10.18.1.1 Directory containing many files	748
10.18.1.2 Maximum Number of Open Files	748
10.18.1.2.1 Maximum number of open files for the entire system	749
10.18.1.2.2 Maximum number of open files per process	749
11 Extras de PostGIS	752
11.1 Normalizador de Direcciones	752
11.1.1 Cómo funciona el analizador	752
11.1.2 Tipos de Address Standardizer	753
11.1.2.1 stdaddr	753
11.1.3 Tipos de Address Standardizer	753
11.1.3.1 rules table	753
11.1.3.2 lex table	756
11.1.3.3 gaz table	756
11.1.4 Funciones de Address Standardizer	757
11.1.4.1 debug_standardize_address	757
11.1.4.2 parse_address	759
11.1.4.3 standardize_address	760
11.2 Geocodificador Tiger	761
11.2.1 Drop_Indexes_Generate_Script	762
11.2.2 Drop_Nation_Tables_Generate_Script	763
11.2.3 Drop_State_Tables_Generate_Script	763
11.2.4 Geocode	764
11.2.5 Geocode_Intersection	766
11.2.6 Get_Geocode_Setting	767
11.2.7 Get_Tract	768
11.2.8 Install_Missing_Indexes	769
11.2.9 Loader_Generate_Census_Script	770
11.2.10 Loader_Generate_Script	772
11.2.11 Loader_Generate_Nation_Script	773
11.2.12 Missing_Indexes_Generate_Script	774
11.2.13 Normalize_Address	775
11.2.14 Pagc_Normalize_Address	776
11.2.15 Pprint_Addy	778
11.2.16 Reverse_Geocode	779
11.2.17 Topology_Load_Tiger	781
11.2.18 Set_Geocode_Setting	783

12 PostGIS Special Functions Index	784
12.1 PostGIS Aggregate Functions	784
12.2 PostGIS Window Functions	785
12.3 PostGIS SQL-MM Compliant Functions	785
12.4 PostGIS Geography Support Functions	801
12.5 PostGIS Raster Support Functions	802
12.6 PostGIS Geometry / Geography / Raster Dump Functions	807
12.7 PostGIS Box Functions	808
12.8 PostGIS Functions that support 3D	809
12.9 PostGIS Curved Geometry Support Functions	814
12.10 PostGIS Polyhedral Surface Support Functions	817
12.11 PostGIS Function Support Matrix	820
12.12 New, Enhanced or changed PostGIS Functions	831
12.12.1 PostGIS Functions new or enhanced in 3.4	831
12.12.2 PostGIS Functions new or enhanced in 3.3	832
12.12.3 PostGIS Functions new or enhanced in 3.2	833
12.12.4 PostGIS Functions new or enhanced in 3.1	833
12.12.5 PostGIS Functions new or enhanced in 3.0	834
12.12.6 PostGIS Functions new or enhanced in 2.5	836
12.12.7 PostGIS Functions new or enhanced in 2.4	837
12.12.8 PostGIS Functions new or enhanced in 2.3	838
12.12.9 PostGIS Functions new or enhanced in 2.2	839
12.12.10 PostGIS Functions new or enhanced in 2.1	841
12.12.11 PostGIS Functions new or enhanced in 2.0	843
12.12.12 PostGIS Functions new or enhanced in 1.5	848
12.12.13 PostGIS Functions new or enhanced in 1.4	850
12.12.14 PostGIS Functions new or enhanced in 1.3	850
13 Informar de problemas	851
13.1 Informar sobre errores de software	851
13.2 Informando sobre problemas de documentación	851
A Apéndice	853
A.1 PostGIS 3.4.0beta1	853
A.1.1 New features	853
A.1.2 Mejoras	854
A.1.3 Breaking Changes	854

Abstract

PostGIS es una extension del sistema de base de datos relacional **PostgreSQL** que permite almacenar objetos SIG (Sistemas de Información Geografica) en la base de datos. PostGIS incluye soporte de indices de tipos basados en GiST R-Tree, y funciones de análisis y procesado de objetos SIG.



Este es el manual de la version 3.4.0beta1



This work is licensed under a **Creative Commons Attribution-Share Alike 3.0 License**. Feel free to use this material any way you like, but we ask that you attribute credit to the PostGIS Project and wherever possible, a link back to <https://postgis.net>.

Chapter 1

Introducción

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

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

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

1.1 Comité de Dirección del Proyecto (Project Steering Committee)

El Comité de Dirección del Proyecto PostGIS (PSC por sus siglas en Ingles, Project Steering Committee) coordina la dirección general, ciclos de publicación, documentación, y el alcance de los esfuerzos para el proyecto PostGIS. Además el PSC da soporte general a usuarios, acepta y aprueba los parches de la comunidad general de PostGIS y vota sobre diversos asuntos relacionados con PostGIS como el acceso de nuevos desarrolladores, los nuevos miembros del PSC o cambios importantes en el API.

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

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

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

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

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

1.2 Core Contributors Present

Nicklas Avén Distance function enhancements (including 3D distance and relationship functions) and additions, Tiny WKB (TWKB) output format and general user support

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

Martin Davis GEOS enhancements and documentation

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

Aliaksandr Kalenik Geometry Processing, PostgreSQL gist, general bug fixing

1.3 Core Contributors Past

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

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

Jorge Arévalo Desarrollo raster, soporte del driver GDAL, cargador

Olivier Courtin Entrada y salida XML (KML,GML)/Funciones GeoJSON, soporte 3D y corrección de errores.

Chris Hodgson Anterior miembro del PSC. Desarrollo en general, mantenimiento del sitio web y buildbot, gestor de la incubación en el OSGeo

Mateusz Loskot CMake support for PostGIS, built original raster loader in python and low level raster API functions

Kevin Neufeld Prior PSC Member. Documentation and documentation support tools, buildbot maintenance, advanced user support on PostGIS newsgroup, and PostGIS maintenance function enhancements.

Dave Blasby El desarrollado/Cofundador de PostGIS original. Dave escribió el código de los objetos del lado del servidor, enlaces de los índices y muchas otras funciones analíticas del lado del servidor.

Jeff Lounsbury Desarrollo original del cargador/descargador de ficheros Shape. Es el propietario representativo actual del proyecto PostGIS.

Mark Leslie Mantenimiento y desarrollo de funciones básicas. Soporte de la curva de mejora. Cargador de Shapefiles.

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

David Zwarg Raster development (mostly map algebra analytic functions)

1.4 Other Contributors

Individual Contributors

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

Corporate Sponsors Estas son compañías que han contribuido con tiempo de desarrollo, alojamiento o con aportes de capital económico al proyecto PostGIS

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

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

Campañas de Crowd Funding Crowd funding campaigns are campaigns we run to get badly wanted features funded that can service a large number of people. Each campaign is specifically focused on a particular feature or set of features. Each sponsor chips in a small fraction of the needed funding and with enough people/organizations contributing, we have the funds to pay for the work that will help many. If you have an idea for a feature you think many others would be willing to co-fund, please post to the [PostGIS newsgroup](#) your thoughts and together we can make it happen.

PostGIS 2.0.0 fue la primer version en la que utilizamos esta estrategia. Utilizamos [PledgeBank](#) y hemos tenido dos campañas con éxito para realizarlas.

postgistopology - 10 patrocinadores contribuyeron con \$ 250 USD cada uno para construir la función toTopoGeometry y con este apoyo, topología 2.0.0. Sucedió.

postgistopology - 10 patrocinadores contribuyeron con \$ 250 USD cada uno para construir la función toTopoGeometry y con este apoyo, topología 2.0.0. Sucedió.

Librerías de soporte importantes The [GEOS](#) geometry operations library

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

The [PROJ](#) cartographic projection library

Por ultimo pero no menos importante, [PostgreSQL DBMS](#), El Gigante sobre el que se apoya PostGIS. Gran parte de la velocidad y la flexibilidad de PostGIS no sería viable sin la extensibilidad, gran planeador de consultas, el índice de GIST, y gran cantidad de funciones SQL que ofrece PostgreSQL.

Chapter 2

Instalación de PostGIS

En este capítulo se detallan los pasos necesarios para instalar PostGIS .

2.1 Versión corta

Para compilar asumiendo que tiene todas las dependencias en su ruta de búsqueda:

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

Una vez que se instala PostGIS, es necesario habilitarlo en cada base de datos en la que desee utilizarlo.

2.2 Compilación e instalación desde el código fuente

Note

Muchos Sistemas Operativos incluyen ya paquetes precompilados PostgreSQL/PostGIS. En la mayoría de casos no es necesario compilar salvo si quieres las ultimas versiones o haces mantenimiento de paquetes.



This section includes general compilation instructions, if you are compiling for Windows etc or another OS, you may find additional more detailed help at [PostGIS User contributed compile guides](#) and [PostGIS Dev Wiki](#).

Pre-Built Packages for various OS are listed in [PostGIS Pre-built Packages](#)

Si eres usuario de Windows, puedes obtener versiones estables compiladas via Stackbuilder o el [sitio de descargas de PostGIS para Windows](#) También tenemos [versiones experimentales para Windows](#) que son publicadas normalmente una o dos veces por semana o cuando ocurre algo interesante. Puedes utilizar éstas para experimentar con las versiones de desarrollo de PostGIS

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

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

Note

Para tener compatibilidad con las funcionalidades GEOS, cuando instales PostgreSQL necesitar hacer un enlace explícito con la librería estándar C++:



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

Esta es una solución para falsas excepciones C++ con herramientas de desarrollo antiguas. Prueba este truco si experimentas problemas extraños (backend cerrado inesperadamente o cosas similares). Por supuesto, será necesario volver a compilar PostgreSQL desde cero.

Los siguientes pasos describen la configuración y compilación del código fuente de PostGIS. Están escritas para usuarios Linux y no funcionarán con Windows o Mac.

2.2.1 Obteniendo el código fuente

Retrieve the PostGIS source archive from the downloads website <http://download.osgeo.org/postgis/source/postgis-3.4.0beta1.tar.gz>

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

Esto creará un directorio llamado `postgis-3.4.0beta1` en el directorio de trabajo actual.

De forma alternativa se puede obtener el código fuente del [git](https://git.osgeo.org/gitea/postgis/postgis/) repositorio <https://git.osgeo.org/gitea/postgis/postgis/>.

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

Vaya al nuevo directorio `postgis-3.4.0beta1` creado para continuar con la instalación.

```
./configure
```

2.2.2 Install Requirements

PostGIS tiene los siguientes requisitos para compilarlo y usarlo:

Requerido

- PostgreSQL 12 - 16. A complete installation of PostgreSQL (including server headers) is required. PostgreSQL is available from <http://www.postgresql.org>.
For a full PostgreSQL / PostGIS support matrix and PostGIS/GEOS support matrix refer to <https://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS>
- compilador GNU C (`gcc`). Otros compiladores ANSI C pueden utilizarse para compilar PostGIS, pero encontraremos menos problemas al compilar con `gcc`.
- GNU Make (`gmake` or `make`). Para muchos sistemas, GNU `make` es la versión por defecto de `make`. Para verificar la versión de `make` podemos ejecutar el siguiente comando `make -v`. Otras versiones de `make` pueden no procesar el fichero PostGIS `Makefile` de forma correcta.
- Proj reprojection library. Proj 6.1 or above is required. The Proj library is used to provide coordinate reprojection support within PostGIS. Proj is available for download from <https://proj.org/>.
- GEOS geometry library, version 3.6 or greater, but GEOS 3.11+ is required to take full advantage of all the new functions and features. GEOS is available for download from <https://libgeos.org>.

- LibXML2, version 2.5.x or higher. LibXML2 is currently used in some imports functions (ST_GeomFromGML and ST_GeomFromKML). LibXML2 is available for download from <https://gitlab.gnome.org/GNOME/libxml2/-/releases>.
- JSON-C, version 0.9 or higher. JSON-C is currently used to import GeoJSON via the function ST_GeomFromGeoJson. JSON-C is available for download from <https://github.com/json-c/json-c/releases/>.
- GDAL, version 2+ is required 3+ is preferred. This is required for raster support. <https://gdal.org/download.html>.
- If compiling with PostgreSQL+JIT, LLVM version >=6 is required <https://trac.osgeo.org/postgis/ticket/4125>.

Optional

- GDAL (pseudo optional) only if you don't want raster you can leave it out. Also make sure to enable the drivers you want to use as described in Section 3.2.
- GTK (GTK+2.0, 2.8+ requerida) para compilar el cargador de shapefiles shp2pgsql-gui. <http://www.gtk.org/> .
- SFCGAL, version 1.3.1 (or higher), 1.4.1 or higher is recommended and required to be able to use all functionality. SFCGAL can be used to provide additional 2D and 3D advanced analysis functions to PostGIS cf Section 7.21. And also allow to use SFCGAL rather than GEOS for some 2D functions provided by both backends (like ST_Intersection or ST_Area, for instance). A PostgreSQL configuration variable `postgis.backend` allow end user to control which backend he want to use if SFCGAL is installed (GEOS by default). Nota: SFCGAL 1.2 require at least CGAL 4.3 and Boost 1.54 (cf: <https://oslandia.gitlab.io/SFCGAL/dev.html>) <https://gitlab.com/Oslandia/SFCGAL/>.
- In order to build the Section 11.1 you will also need PCRE <http://www.pcre.org> (which generally is already installed on nix systems). Section 11.1 will automatically be built if it detects a PCRE library, or you pass in a valid `--with-pcre-dir=/path/to/pcre` during configure.
- To enable ST_AsMVT protobuf-c library 1.1.0 or higher (for usage) and the protoc-c compiler (for building) are required. Also, pkg-config is required to verify the correct minimum version of protobuf-c. See [protobuf-c](#). By default, Postgis will use Wagyuu to validate MVT polygons faster which requires a c++11 compiler. It will use CXXFLAGS and the same compiler as the PostgreSQL installation. To disable this and use GEOS instead use the `--without-wagyuu` during the configure step.
- CUnit (CUnit). Se necesita para hacer test de regresión. <http://cunit.sourceforge.net/>
- DocBook (xsltproc) es necesario para compilar la documentación. Docbook esta disponible en <http://www.docbook.org/> .
- DBLatex (dbratex) necesario para construir la documentación en formato PDF. DBLatex esta disponible en <http://dblatex.sourceforge.net/> .
- ImageMagick (convert) es necesario para generar las imágenes empleadas en la documentación. ImageMagick esta disponible en <http://www.imagemagick.org/> .

2.2.3 Configuración

Como en la gran mayoría de instalaciones Linux, el primer paso es generar el Makefile que se utilizara para compilar el código fuente. Esto se hace ejecutando el script de shell.

./configure

Sin parámetros adicionales, este comando intentara localizar los componentes y librerías necesarios para construir el código fuente PostGIS de forma automática en tu sistema. Aunque este es el uso mas común de **./configure**, el script acepta varios parámetros para aquellos que han instalado las librerías y programas en lugares no standard.

La siguiente lista muestra los parámetros utilizados mas comunes. Para obtener una lista completa, puedes utilizar los parámetros **--help** o **--help=short**.

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

--prefix=PREFIX Esta es la localización donde se instalarán las librerías PostGIS y los scripts SQL. Por defecto, esta localización es la misma que la detectada para la instalación PostgreSQL.



Caution

Este parámetro está roto actualmente, ya que el paquete sólo se instalará en el directorio de instalación de PostgreSQL. Para seguir al tanto de este bug visita <http://trac.osgeo.org/postgis/ticket/635>

--with-pgconfig=FILE PostgreSQL tiene una herramienta llamada **pg_config** para activar extensiones como PostGIS o para localizar el directorio de instalación de PostgreSQL. Utiliza este parámetro (**--with-pgconfig=/path/to/pg_config**) para especificar una instalación personalizada de PostgreSQL de forma manual que PostGIS utilizara para compilar.

--with-gdalconfig=FILE GDAL, una biblioteca necesaria, proporciona la funcionalidad necesaria para el soporte de raster **gdal-config** para activar la instalación de software para localizar el directorio de instalación de GDAL. Utilice este parámetro (**- with-gdalconfig = / ruta /a/ gdal config-**) para especificar manualmente una instalación de GDAL personalizada que PostGIS utilizara para compilar.

--with-geosconfig=FILE GEOS, librería de geometrías requerida, tiene una utilidad llamada **geos-config** para activar la localización del directorio de instalación del software GEOS. Utiliza este parámetro (**--with-geosconfig=/path/to/geos-config**) para especificar de forma manual una instalación personalizada de GEOS que PostGIS puede utilizar para compilar.

--with-xml2config=FILE LibXML es una librería necesaria para procesar GeomFromKML/GML. Normalmente encontrará si tienes instalada la librería libxml, pero si no está instalada, o quieres usar una versión específica, necesitarás que PostGIS apunte a un fichero de configuración particular **xml2-config** para localizar un directorio de instalación LibXML para activar la instalación del Software. Utiliza el siguiente parámetro (**>--with-xml2config=/path/to/xml2-config**) para especificar de forma manual una instalación personalizada de LibXML con la que compilar PostGIS.

--with-projdir=DIR Proj4 es una librería de reproyecciones necesaria de PostGIS. Utiliza el siguiente parámetro (**--with-projdir=/path/to/projdir**) para definir manualmente una instalación personalizada de Proj4 para compilar PostGIS.

--with-libiconv=DIR Directorio donde iconv está instalado.

--with-jsondir=DIR **JSON-C** es una librería con licencia MIT-licensed JSON necesaria para dar soporte a PostGIS ST_GeomFromJSON. Utiliza este parámetro (**--with-jsondir=/path/to/jsondir**) para especificar de forma manual el directorio de instalación personalizado de instalación de JSON-C que PostGIS utilizara para compilar.

--with-pcre=DIR **PCRE** is an BSD-licensed Perl Compatible Regular Expression library required by address_standardizer extension. Use this parameter (**--with-pcre=dir**) to manually specify a particular PCRE installation directory that PostGIS will build against.

--with-gui Compilar la GUI de importar datos (necesita GTK+2.0). Esto creará una interfaz gráfica **shp2pgsql-gui** para el comando **shp2pgsql**.

--without-raster Compile without raster support.

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

--with-gettext=no PostGIS intentará detectar soporte gettext y compilar con él por defecto, de todas formas si existen incompatibilidades que causan errores de carga, se puede desactivar por completo con este comando. Para ver un ejemplo de resolución de problemas configurando en gettext puedes ver el siguiente enlace <http://trac.osgeo.org/postgis/ticket/748>. NOTA: No te pierdes mucho si desactivas esta opción. Se utiliza principalmente para soporte de ayuda/etiquetas internacionales en la GUI de carga, que actualmente no está documentada y sigue siendo experimental.

--with-sfcgal=PATH By default PostGIS will not install with sfcgal support without this switch. PATH is an optional argument that allows to specify an alternate PATH to sfcgal-config.

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

**Note**

If you obtained PostGIS from the [code repository](#), the first step is really to run the script

./autogen.sh

Este Script generara el script **configure** que a su vez se utiliza para personalizar la instalación de PostGIS.

Si, por el contrario, as obtenido PostGIS como tarball, ejecutar **./autogen.sh** no es necesario ya que ya se ha generado **configure**.

2.2.4 Compilando

Una vez generado el Makefile, compilar PostGIS es tan simple como ejecutar

make

La ultima linea de salida del terminal debe ser "PostGIS copilado con éxito. Listo para instalar."

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

make comments

Introducido en la version PostGIS 2.0. Esto genera hojas de referencia html para una referencia rápida o para los folletos. Esto requiere xsltproc para compilar y generará 4 ficheros en la carpeta doc `topology_cheatsheet.html`, `tiger_geocoder_cheatsheet.html`, `raster_cheatsheet.html`, `postgis_cheatsheet.html`

Puedes descargar algunos ya compilados en formato html o pdf en [Guías de Estudio PostGIS / PostgreSQL](#)

make cheatsheets

2.2.5 Compilando e Instalando Extensiones de PostGIS

Las extensiones de PosGIS son compiladas e instaladas de forma automatica si estas utilizando la version 9.1+ de PostgreSQL

Si estas compilando desde el repositorio de código fuente, necesitas compilar primero la función descriptions. Si tienes instalado docbook ya esta compilado. También puedes compilarla manualmente con la sentencia:

make comments

Compilar los comentarios no es necesario si estas compilando desde un tar ya que están en el paquete pre-compilados con el tar.

Si estas compilando para PostgreSQL 9.1, la extension debería compilarse de forma automática como parte del proceso del comando `make install`. Si lo necesitas, puedes compilar la extensión desde las carpetas de la extensión o copiar los ficheros en un servidor diferente.

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

**Note**

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

Los ficheros de la extension serán siempre los mismos para la misma versión de PostgreSQL independientemente del Sistema Operativo, así que se pueden copiar los ficheros de la extensión de un Sistema Operativo a otro si ya tienes los binarios de PostGIS ya instalados en tus servidores.

Si quieres instalar la extensión de forma manual en un servidor separado de tu servidor de desarrollo, necesitas copiar los siguientes archivos de la carpeta de la extensión en la carpeta PostgreSQL / share / extension de la instalación de PostgreSQL y los binarios normales para PostGIS si no los tienes instalados en el servidor.

- Estos son los ficheros de control que contienen información como la versión de la extensión a instalar si no lo has especificado. `postgis.control`, `postgis_topology.control`.
- Todos los ficheros en la carpeta `/sql` de la extension. Estos ficheros deben ser copiados en la raíz de PostgreSQL en la carpeta `share/extension extensions/postgis/sql/*.sql, extensions/postgis_topology/sql/*.sql`

Una vez hecho esto deberías ver `postgis`, `postgis_topology` como extensiones disponibles en PgAdmin -> extensiones.

Si estas utilizando `psql`, puedes verificar que las extensiones están instaladas ejecutando la siguiente sentencia:

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

name	default_version	installed_version
address_standardizer	3.4.0beta1	3.4.0beta1
address_standardizer_data_us	3.4.0beta1	3.4.0beta1
postgis	3.4.0beta1	3.4.0beta1
postgis_raster	3.4.0beta1	3.4.0beta1
postgis_sfcgal	3.4.0beta1	
postgis_tiger_geocoder	3.4.0beta1	3.4.0beta1
postgis_topology	3.4.0beta1	

(6 rows)

Si tienes instalada una extension en la base de datos que estas consultando, deberías verla mencionada la columna `installed_version`.

Si la consulta no devuelve ningún registro, significa que no tienes la extension PostGIS instalada en el servidor. PgAdmin III 1.14+ muestra esta información en la sección `extensiones` en el navegador de bases de datos y permite actualizar o instalar haciendo click derecho.

Si la extension esta disponible, puedes instalar la extension `postgis` en la base de datos de tu elección utilizando la interfaz de extensiones de pgAdmin o ejecutando la siguiente sentencia:

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

In `psql` you can use to see what versions you have installed and also what schema they are installed.

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

```
List of installed extensions
-[ RECORD 1 ]-----
Name          | postgis
Version       | 3.4.0beta1
Schema        | public
Description   | PostGIS geometry, geography, and raster spat..
```

```

-[ RECORD 2 ]-----
Name          | postgis_raster
Version       | 3.0.0dev
Schema        | public
Description   | PostGIS raster types and functions
-[ RECORD 3 ]-----
Name          | postgis_tiger_geocoder
Version       | 3.4.0beta1
Schema        | tiger
Description   | PostGIS tiger geocoder and reverse geocoder
-[ RECORD 4 ]-----
Name          | postgis_topology
Version       | 3.4.0beta1
Schema        | topology
Description   | PostGIS topology spatial types and functions

```

Warning

No se pueden hacer copias de seguridad explícitas de las tablas de las extensiones `spatial_ref_sys`, `layer`, `topology`. Solo se pueden hacer copias de seguridad explícitas cuando cuando se hacen copias de seguridad de sus respectivas extensiones `postgis` or `postgis_topology`, lo que al parecer ocurre cuando haces una copia de seguridad de la base de datos completa. Con PostGIS 2.0.1, solo los srid no incluidos en PostGIS son guardados cuando se hace una copia de seguridad de la base de datos, así que no esperes que al cambiar alguno de los srid que incluye PostGIS este en tu copia de seguridad. Envía un ticket si encuentras algún problema. La estructura de las tablas de extensiones no se guardan en copias de seguridad si son creadas con `CREATE EXTENSION` y son la misma estructura para una version dada de una extension. Estos comportamientos están incorporados en el modelo de extensiones PostgreSQL actual, así que nada podemos hacer al respecto.

If you installed 3.4.0beta1, without using our wonderful extension system, you can change it to be extension based by running the below commands to package the functions in their respective extension. Installing using ``unpacked`` was removed in PostgreSQL 13, so you are advised to switch to an extension build before upgrading to PostgreSQL 13.

```

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

```

2.2.6 Tests

Si quieres hacer un test en la compilación de PostGIS, ejecuta

make check

El comando anterior ejecutará varias comprobaciones y tests de regresión utilizando la librería generada para la version de base de datos PostgreSQL actual.

**Note**

Si has configurado PostGIS con instalaciones de PostgreSQL, GEOS, o Proj4 en directorios personalizados, necesitaras añadir las localizaciones de las librerías personalizadas en la variable de entorno `LD_LIBRARY_PATH`.

**Caution**

Actualmente, el comando **make check** une las variables de entorno `PATH` y `PGPORT` cundo ejecuta las comprobaciones - *no* utiliza la version de PostgreSQL especificada utilizando el parametro de configuración **--with-pgconfig**. Así que hay que estar seguros de modificar la variable de entorno `PATH` para que apunte a la instalación de PostgreSQL detectada durante la configuración o estar preparado para tener algún que otro dolor de cabeza.

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

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

.
.
.

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

.
.
.

Running tests

.
.
.

Run tests: 134
Failed: 0

-- if you build with SFCGAL

.
.
.

Running tests

.
.
.

Run tests: 13
Failed: 0

-- if you built with raster support

.
.
.

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

.
.
.

Running tests
```

```

    .
    .
    .

Run tests: 101
Failed: 0

-- topology regress

.
.
.

Running tests

    .
    .
    .

Run tests: 51
Failed: 0

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

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

    .
    .
    .

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

```

The `postgis_tiger_geocoder` and `address_standardizer` extensions, currently only support the standard PostgreSQL `installcheck`. To test these use the below. Note: the `make install` is not necessary if you already did `make install` at root of PostGIS code folder.

For `address_standardizer`:

```

cd extensions/address_standardizer
make install
make installcheck

```

Output should look like:

```

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

=====
All 4 tests passed.

```

For tiger geocoder, make sure you have postgis and fuzzystrmatch extensions available in your PostgreSQL instance. The address_standardizer tests will also kick in if you built postgis with address_standardizer support:

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

output should look like:

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

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

2.2.7 Instalación

Para instalar PostGIS entre

make install

Esto copiará los ficheros de instalación de PostGIS en el subdirectorio especificado por el parámetro de configuración **--prefix** del comando . En particular:

- Los archivos binarios de carga y dumper estarán instalados en [prefix]/bin.
- Los archivos SQL, tal como postgis.sql, están instalados en [prefix]/share/contrib.
- Las librerías de PostGIS estarán instaladas en [prefix]/lib.

Si has ejecutado el comando **make comments** previamente para generar los ficheros postgis_comments.sql, raster_comments.sql, instala los ficheros sql ejecutando:

make comments-install



Note

postgis_comments.sql, raster_comments.sql, topology_comments.sql han sido separados de la compilación y de la instalación típicos ya que tienen una dependencia extra de la librería **xsltproc**.

2.3 Installing and Using the address standardizer

The `address_standardizer` extension used to be a separate package that required separate download. From PostGIS 2.2 on, it is now bundled in. For more information about the `address_standardize`, what it does, and how to configure it for your needs, refer to Section 11.1.

This standardizer can be used in conjunction with the PostGIS packaged tiger geocoder extension as a replacement for the `Normalize_Address` discussed. To use as replacement refer to Section 2.4.2. You can also use it as a building block for your own geocoder or use it to standardize your addresses for easier compare of addresses.

The address standardizer relies on PCRE which is usually already installed on many Nix systems, but you can download the latest at: <http://www.pcre.org>. If during Section 2.2.3, PCRE is found, then the address standardizer extension will automatically be built. If you have a custom pcre install you want to use instead, pass to configure `--with-pcredir=/path/to/pcre` where `/path/to/pcre` is the root folder for your pcre include and lib directories.

For Windows users, the PostGIS 2.1+ bundle is packaged with the `address_standardizer` already so no need to compile and can move straight to CREATE EXTENSION step.

Once you have installed, you can connect to your database and run the SQL:

```
CREATE EXTENSION address_standardizer;
```

The following test requires no rules, gaz, or lex tables

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

Output should be

num	street	city	state	zip
1	Devonshire Place PH301	Boston	MA	02109

2.4 Installing, Upgrading Tiger Geocoder, and loading data

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

2.4.1 Tiger Geocoder Enabling your PostGIS database

1. These directions assume your PostgreSQL installation already has the `postgis_tiger_geocoder` extension installed.
2. Connect to your database via psql or pgAdmin or some other tool and run the following SQL commands. Note that if you are installing in a database that already has `postgis`, you don't need to do the first step. If you have `fuzzystrmatch` extension already installed, you don't need to do the second step either.

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

If you already have `postgis_tiger_geocoder` extension installed, and just want to update to the latest run:

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


If you made custom entries or changes to `tiger.loader_platform` and `tiger.loader_variables` you may need to update these.

3. To confirm your install is working correctly, run this sql in your database:

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

Which should output

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

4. Create a new record in `tiger.loader_platform` table with the paths of your executables and server.

So for example to create a profile called `debbie` that follows `sh` convention. You would do:

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

And then edit the paths in the `declare_sect` column to those that fit Debbie's `pg`, `unzip`, `shp2pgsql`, `psql`, etc path locations.

If you don't edit this `loader_platform` table, it will just contain common case locations of items and you'll have to edit the generated script after the script is generated.

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

To enable it, do the following:

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

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

6. Create a folder called `gisdata` on root of server or your local pc if you have a fast network connection to the server. This folder is where the tiger files will be downloaded to and processed. If you are not happy with having the folder on the root of the server, or simply want to change to a different folder for staging, then edit the field `staging_fold` in the `tiger.loader_variables` table.
7. Create a folder called `temp` in the `gisdata` folder or wherever you designated the `staging_fold` to be. This will be the folder where the loader extracts the downloaded tiger data.
8. Then run the **Loader_Generate_Nation_Script** SQL function make sure to use the name of your custom profile and copy the script to a `.sh` or `.bat` file. So for example to build the nation load:

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

9. Run the generated nation load commandline scripts.

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

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

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

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

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

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

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

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

Alternatively you can load just these tables after loading state data using the [Loader_Generate_Census_Script](#)

12. For each state you want to load data for, generate a state script [Loader_Generate_Script](#).



Warning

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

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

14. Run the generated commandline scripts.

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

15. After you are done loading all data or at a stopping point, it's a good idea to analyze all the tiger tables to update the stats (include inherited stats)

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

## 2.4.2 Using Address Standardizer Extension with Tiger geocoder

One of the many complaints of folks is the address normalizer function `Normalize_Address` function that normalizes an address for prepping before geocoding. The normalizer is far from perfect and trying to patch its imperfectness takes a vast amount of resources. As such we have integrated with another project that has a much better address standardizer engine. To use this new `address_standardizer`, you compile the extension as described in Section 2.3 and install as an extension in your database.

Once you install this extension in the same database as you have installed `postgis_tiger_geocoder`, then the `Pagc_Normalize_Address` can be used instead of `Normalize_Address`. This extension is tiger agnostic, so can be used with other data sources such as international addresses. The tiger geocoder extension does come packaged with its own custom versions of `rules table` (`tiger.pagc_rules`), `gaz table` (`tiger.pagc_gaz`), and `lex table` (`tiger.pagc_lex`). These you can add and update to improve your standardizing experience for your own needs.

## 2.4.3 Required tools for tiger data loading

El proceso de carga, descarga datos desde el sitio web del censo de las respectivas naciones de los estados pedidos, extrae los ficheros, y carga cada estado en un conjunto separado por estados en su propia tabla. Cada tabla de estado hereda el esquema de tablas definido en `tiger` así que basta con hacer una consulta a estas tablas para acceder a todos los datos de la tabla de estados en cualquier momento utilizando `Drop_State_Tables_Generate_Script` si necesita volver a cargar un estado o si ya no lo necesitas mas.

Para poder cargar los datos necesitarás las siguientes herramientas:

- Una herramienta para descomprimir ficheros zip de la pagina web del censo.  
Para sistemas Unix: el ejecutable `unzip` que normalmente esta instalado en la mayoría de sistemas Unix.  
Para windows, 7-zip es una herramienta libre de compresión/descompresión que puedes descargar en <http://www.7-zip.org/>
- El comando `shp2pgsql` que se instala por defecto cuando instalas PostGIS.
- `wget` que es una herramienta de captura web, normalmente instalado en los sistemas Unix/Linux.  
Si estas en windows, puedes obtener ejecutables precompilados en <http://gnuwin32.sourceforge.net/packages/wget.htm>

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

To load state data refer to `Loader_Generate_Script` to generate a data load script for your platform for the states you desire. Note that you can install these piecemeal. You don't have to load all the states you want all at once. You can load them as you need them.

Una vez que los estados que quieres han sido cargados, asegure de ejecutar:

```
SELECT install_missing_indexes();
```

como se explica en `Install_Missing_Indexes`.

Para probar que las cosas han funcionado como deberían, intenta ejecutar una geocodificación en una dirección del estado descargado utilizando `Geocode`

## 2.4.4 Upgrading your Tiger Geocoder Install and Data

First upgrade your `postgis_tiger_geocoder` extension as follows:

```
ALTER EXTENSION postgis_tiger_geocoder UPDATE;
```

Después, elimina todas las tablas de naciones y carga las nuevas. Genera un script drop con esta sentencia SQL como se detalla en `Drop_Nation_Tables_Generate_Script`

```
SELECT drop_nation_tables_generate_script();
```

Ejecuta la sentencia SQL drop

Genera un script de carga de naciones con esta sentencia SELECT como se detalla en [Loader\\_Generate\\_Nation\\_Script](#)

#### Para windows

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

#### Para unix/linux

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

Refer to Section [2.4.1](#) for instructions on how to run the generate script. This only needs to be done once.



#### Note

You can have a mix of different year state tables and can upgrade each state separately. Before you upgrade a state you first need to drop the prior year state tables for that state using [Drop\\_State\\_Tables\\_Generate\\_Script](#).

## 2.5 Common Problems during installation

Hay varias cosas a comprobar cuando la instalación o actualización no han fusionado como se esperaba.

1. Comprueba que tienes instalado PostgreSQL 12 o posterior, y que estas compilando para esta version de PostgreSQL que estas utilizando. Se pueden producir confusiones cuando tu distribución (Linux) ya tiene instalada PostgreSQL, o has instalado antes PostgreSQL y lo has olvidado. PostGIS solo funcionará con PostgreSQL 12 o superior, y errores inesperados o extraños pueden ocurrir si utilizas una version mas antigua. Para comprobar la version de PostgreSQL que esta instalada y ejecutándose, conectare a la base de datos utilizando psql y ejecuta la siguiente consulta:

```
SELECT version();
```

Si estas ejecutando una version basada en una distribución RPM, puedes comprobar si existen paquetes pre-instalados utilizando el comando **rpm** como sigue: **rpm -qa | grep postgresql**

2. Si tienes errores en la actualización, asegúrate de que estas restaurando tu base de datos en una que tenga instalada PostGIS.

```
SELECT postgis_full_version();
```

Comprueba que tu configuración detecta la ubicación y la version correctas de PostgreSQL, la librería Proj4 y la librería GEOS.

1. La salida de configure se utiliza para generar el fichero `postgis_config.h`. Comprueba que la variables `POSTGIS_PGSQL_VERSION` y `POSTGIS_PROJ_VERSION` y `POSTGIS_GEOS_VERSION`, han sido bien configuradas.

## Chapter 3

# Administración de PostGIS

### 3.1 Performance Tuning

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

For general details about optimizing PostgreSQL, refer to [Tuning your PostgreSQL Server](#).

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

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

In addition to the Postgres settings, PostGIS has some custom settings which are listed in [Section 7.24](#).

#### 3.1.1 Startup

These settings are configured in `postgresql.conf`:

##### `constraint_exclusion`

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

##### `shared_buffers`

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

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

- Predeterminado: 18
  - Sets the maximum number of background processes that the system can support. This parameter can only be set at server start.
-

### 3.1.2 Runtime

**work\_mem** - sets the size of memory used for sort operations and complex queries

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

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

**maintenance\_work\_mem** - the memory size used for VACUUM, CREATE INDEX, etc.

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

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

**max\_parallel\_workers\_per\_gather**

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

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

## 3.2 Configuring raster support

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

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

If you want to enable offline raster:

```
POSTGIS_ENABLE_OUTDB_RASTERS=1
```

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

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

```
POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

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

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

**Note**

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

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

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

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

## 3.3 Creating spatial databases

### 3.3.1 Spatially enable database using EXTENSION

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

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

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

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

### 3.3.2 Spatially enable database without using EXTENSION (discouraged)

**Note**

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

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

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

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

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

Run the following Shell commands in your terminal:

```

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

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

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

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

```

## 3.4 Upgrading spatial databases

Upgrading existing spatial databases can be tricky as it requires replacement or introduction of new PostGIS object definitions. Unfortunately not all definitions can be easily replaced in a live database, so sometimes your best bet is a dump/reload process. PostGIS provides a SOFT UPGRADE procedure for minor or bugfix releases, and a HARD UPGRADE procedure for major releases.

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

### 3.4.1 Soft upgrade

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

#### 3.4.1.1 Soft Upgrade 9.1+ using extensions

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

If you are running PostGIS 3 or above, then you should use the [PostGIS\\_Extensions\\_Upgrade](#) function to upgrade to the latest version you have installed.

```
SELECT postgis_extensions_upgrade();
```

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

```

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

```

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

```

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

```

If you get an error notice something like:



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

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

If you get a notice message like:

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

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

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



#### Note

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



#### Note

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

```
DROP EXTENSION postgis_raster;
```

### 3.4.1.2 Soft Upgrade Pre 9.1+ or without extensions

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

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

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

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

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

Load them all in turn, starting from postgis\_upgrade.sql.

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

The same procedure applies to raster, topology and sfcgal extensions, with upgrade files named rtpostgis\_upgrade.sql, topology\_upgrade.sql and sfcgal\_upgrade.sql respectively. If you need them:

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

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

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

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

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



#### Note

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

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

### 3.4.2 Hard upgrade

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

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

Supplementary instructions for windows users are available at [Windows Hard upgrade](#).

The Procedure is as follows:

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

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

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

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

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

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

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

Errors may arise in the following cases:

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

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

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

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

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

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

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

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

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

run this command :

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

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

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

## Chapter 4

# Data Management

### 4.1 Spatial Data Model

#### 4.1.1 OGC Geometry

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

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

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

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

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

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

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

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

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

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

#### 4.1.1.1 Point

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

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

#### 4.1.1.2 LineString

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

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

#### 4.1.1.3 LinearRing

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

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

#### 4.1.1.4 Polygon

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

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

#### 4.1.1.5 MultiPoint

A MultiPoint is a collection of Points.

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

#### 4.1.1.6 MultiLineString

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

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

#### 4.1.1.7 MultiPolygon

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

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

#### 4.1.1.8 GeometryCollection

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

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

#### 4.1.1.9 PolyhedralSurface

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

```
POLYHEDRALSURFACE Z (
 ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
 ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
 ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
 ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
 ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
 ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))
```

#### 4.1.1.10 Triangle

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

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

#### 4.1.1.11 TIN

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

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

### 4.1.2 SQL/MM Part 3 - Curves

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



#### Note

Todas las comparaciones de coma flotante en la implementación SQL-MM se desarrollan para una tolerancia específica, normalmente 1E-8.

#### 4.1.2.1 CircularString

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

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

#### 4.1.2.2 CompoundCurve

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

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

#### 4.1.2.3 CurvePolygon

A CurvePolygon is like a polygon, with an outer ring and zero or more inner rings. The difference is that a ring can be a CircularString or CompoundCurve as well as a LineString.

A partir de PostGIS 1.4, PostGIS soporta curvas compuestas en un polígono curvo.

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

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

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

#### 4.1.2.4 MultiCurve

A MultiCurve is a collection of curves which can include LineStrings, CircularStrings or CompoundCurves.

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

#### 4.1.2.5 MultiSurface

A MultiSurface is a collection of surfaces, which can be (linear) Polygons or CurvePolygons.

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

### 4.1.3 WKT and WKB

The OGC SFA specification defines two formats for representing geometry values for external use: Well-Known Text (WKT) and Well-Known Binary (WKB). Both WKT and WKB include information about the type of the object and the coordinates which define it.

Well-Known Text (WKT) provides a standard textual representation of spatial data. Examples of WKT representations of spatial objects are:

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

Input and output of WKT is provided by the functions **ST\_AsText** and **ST\_GeomFromText**:

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

For example, a statement to create and insert a spatial object from WKT and a SRID is:

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

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

- WKT: POINT(1 1)  
WKB: 010100000000000000000000F03F000000000000F03
- WKT: LINESTRING (2 2, 9 9)  
WKB: 010200000002000000000000000000004000000000000000400000000000022400000000000002240

Input and output of WKB is provided by the functions **ST\_AsBinary** and **ST\_GeomFromWKB**:

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

For example, a statement to create and insert a spatial object from WKB is:

```
INSERT INTO geotable (geom, name)
VALUES (ST_GeomFromWKB('\x010100000000000000000000f03f000000000000f03f', 312), 'A Place');
```



## 4.2 Geometry Data Type

PostGIS implements the OGC Simple Features model by defining a PostgreSQL data type called `geometry`. It represents all of the geometry subtypes by using an internal type code (see [GeometryType](#) and [ST\\_GeometryType](#)). This allows modelling spatial features as rows of tables defined with a column of type `geometry`.

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



### Note

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

The SFA standard specifies that spatial objects include a Spatial Reference System identifier (SRID). The SRID is required when creating spatial objects for insertion into the database (it may be defaulted to 0). See [ST\\_SRID](#) and Section 4.5

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

### 4.2.1 PostGIS EWKB and EWKT

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

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

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

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

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

0101000020040000000000000000000000000000000000000000000000000000
```

PostGIS EWKT output has a few differences to OGC WKT:

- For 3DZ geometries the Z qualifier is omitted:  
OGC: POINT Z (1 2 3)  
EWKT: POINT (1 2 3)
- For 3DM geometries the M qualifier is included:  
OGC: POINT M (1 2 3)  
EWKT: POINTM (1 2 3)

- For 4D geometries the ZM qualifier is omitted:

OGC: POINT ZM (1 2 3 4)

EWKT: POINT (1 2 3 4)

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

- POINT ZM (1 1)
- POINT ZM (1 1 1)
- POINT (1 1 1 1)



#### Caution

PostGIS extended formats are currently a superset of the OGC ones, so that every valid OGC WKB/WKT is also valid EWKB/EWKT. However, this might vary in the future, if the OGC extends a format in a way that conflicts with the PostGIS definition. Thus you **SHOULD NOT** rely on this compatibility!

Examples of the EWKT text representation of spatial objects are:

- POINT(0 0 0) -- XYZ
- SRID=32632;POINT(0 0) -- XY with SRID
- POINTM(0 0 0) -- XYM
- POINT(0 0 0 0) -- XYZM
- SRID=4326;MULTIPOINTM(0 0 0,1 2 1) -- XYM with SRID
- MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))
- POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
- MULTIPOLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)),((-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))
- GEOMETRYCOLLECTIONM( POINTM(2 3 9), LINESTRINGM(2 3 4, 3 4 5) )
- MULTICURVE( (0 0, 5 5), CIRCULARSTRING(4 0, 4 4, 8 4) )
- POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )
- TRIANGLE ((0 0, 0 10, 10 0, 0 0))
- TIN( ((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 0 0 0)) )

Input and output using these formats is available using the following functions:

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

For example, a statement to create and insert a PostGIS spatial object using EWKT is:

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

## 4.3 Geography Data Type

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

The basis for the PostGIS geometry data type is a plane. The shortest path between two points on the plane is a straight line. That means functions on geometries (areas, distances, lengths, intersections, etc) are calculated using straight line vectors and cartesian mathematics. This makes them simpler to implement and faster to execute, but also makes them inaccurate for data on the spheroidal surface of the earth.

The PostGIS geography data type is based on a spherical model. The shortest path between two points on the sphere is a great circle arc. Functions on geographies (areas, distances, lengths, intersections, etc) are calculated using arcs on the sphere. By taking the spheroidal shape of the world into account, the functions provide more accurate results.

Because the underlying mathematics is more complicated, there are fewer functions defined for the geography type than for the geometry type. Over time, as new algorithms are added the capabilities of the geography type will expand. As a workaround one can convert back and forth between geometry and geography types.

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

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

### 4.3.1 Creating Geography Tables

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

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

The geography type supports two optional type modifiers:

- the spatial type modifier restricts the kind of shapes and dimensions allowed in the column. Values allowed for the spatial type are: POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION. The geography type does not support curves, TINS, or POLYHEDRALSURFACES. The modifier supports coordinate dimensionality restrictions by adding suffixes: Z, M and ZM. For example, a modifier of 'LINESTRINGM' only allows linestrings with three dimensions, and treats the third dimension as a measure. Similarly, 'POINTZM' requires four dimensional (XYZM) data.
- the SRID modifier restricts the spatial reference system SRID to a particular number. If omitted, the SRID defaults to 4326 (WGS84 geodetic), and all calculations are performed using WGS84.

Examples of creating tables with geography columns:

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

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

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

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

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

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

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

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

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

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

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

```
SELECT * FROM geography_columns;
```

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

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

### 4.3.2 Using Geography Tables

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

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

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

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

0101000020AD100000000000000000C05EC000000000000004140
```

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

0101000020AB100000000000000000C05EC000000000000004140
```

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

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

Las consultas y las funciones de medidas utilizan metros cho unidad. Asi que los parámetros de distancia deben estar expresados en metros, y los valores devueltos deben estar expresados en metros (o metros cuadrados para áreas)

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

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

The geography type calculates the true shortest distance of 122.235 km over the sphere between Reykjavik and the great circle flight path between Seattle and London.

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

122235.23815667
```

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

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

13.342271221453624
```

### 4.3.3 When to use the Geography data type

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

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

- Si tus datos están un área pequeña, la mejor solución seria elegir una proyección adecuada y utilizando GEOMETRY, en términos de rendimiento y funcionalidades disponibles.
- Si tus datos son globales o cubren una región continental, veras que GEOGRAPHY te permite construir un sistema sin tener que preocuparte sobre detalles de proyección. Almacenas tus datos en longitud/latitud, y utilizas las funciones definidas en GEOGRAPHY.
- Si no entiendes las proyecciones, y no quieres aprender sobre ellas, y estas preparado a aceptar las funcionalidades limitadas disponibles en GEOGRAPHY, entonces sera mas fácil para ti, utilizar GEOGRAPHY en lugar de GEOMETRY. Simplemente carga tus datos como longitud/latitud y continua desde allí.

Para tener una comparación entre lo que esta soportado entre Geography y Geometry ve a Section [12.11](#). Para obtener una lista con la descripción de las funciones Geography ve a Section [12.4](#)

### 4.3.4 Preguntas frecuentes Avanzadas de Geography

#### 1. ¿Se calcula en la esfera o en el esferoide?

Por defecto, todos los cálculos de distancia y área están hechos sobre el esferoide. Deberías ver que los resultados de los cálculos en áreas locales deberán coincidir con los resultados en coordenadas locales planas con proyecciones locales correctas. En grandes áreas, los cálculos esferoidales serán mas precisas que cualquier calculo realizado en planas. Todas las funciones "geography" tienen la opción de utilizar el calculo sobre la esfera, seleccionando el parámetro final booleano a 'FALSE'. Esto puede acelerar los cálculos, particularmente en casos donde las geometrias son muy simples.

2. *¿Que ocurre con los husos horarios y los polos?*

Todos los cálculos no tienen nociones de husos horarios o polos, las coordenadas son esféricas(longitud/latitud) así que una forma que atraviesa husos horarios no es, desde un punto de vista de los cálculos, a cualquier otra forma.

3. *¿Cual es el arco mas largo que se puede procesar?*

Utilizamos grandes arcos de círculo como la "línea de interpolación" entre dos puntos. Esto significa que actualmente, dos puntos se unen de dos formas, dependiendo de la dirección del viaje sobre el arco. Todo nuestro código asume que los puntos están unidos por el \*mas corto\* de los dos caminos a través del arco de circunferencia. Como consecuencia, las formas que tienen arcos mayores de 180 grados no serán modeladas correctamente.

4. *¿ Por que es tan lento el calculo del area de Europa / Rusia / añade una región geográfica grande aquí?*

¡Por que el polígono es condenadamente grande! Las grandes áreas son malas por dos razones: Sus límites son grandes, así que el índice tiende a tirar de la función sin importar la consulta que estes ejecutando; el número de vértices es grande, y los tests (distancia, de contención) tiene que recorrer la lista de vértices al menos una vez y a veces N veces ( con N igual al número de vértices en el otro objeto candidato). As with GEOMETRY, we recommend that when you have very large polygons, but are doing queries in small areas, you "denormalize" your geometric data into smaller chunks so that the index can effectively subquery parts of the object and so queries don't have to pull out the whole object every time. Please consult [ST\\_Subdivide](#) function documentation. Just because you \*can\* store all of Europe in one polygon doesn't mean you \*should\*.

## 4.4 Geometry Validation

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

### 4.4.1 Simple Geometry

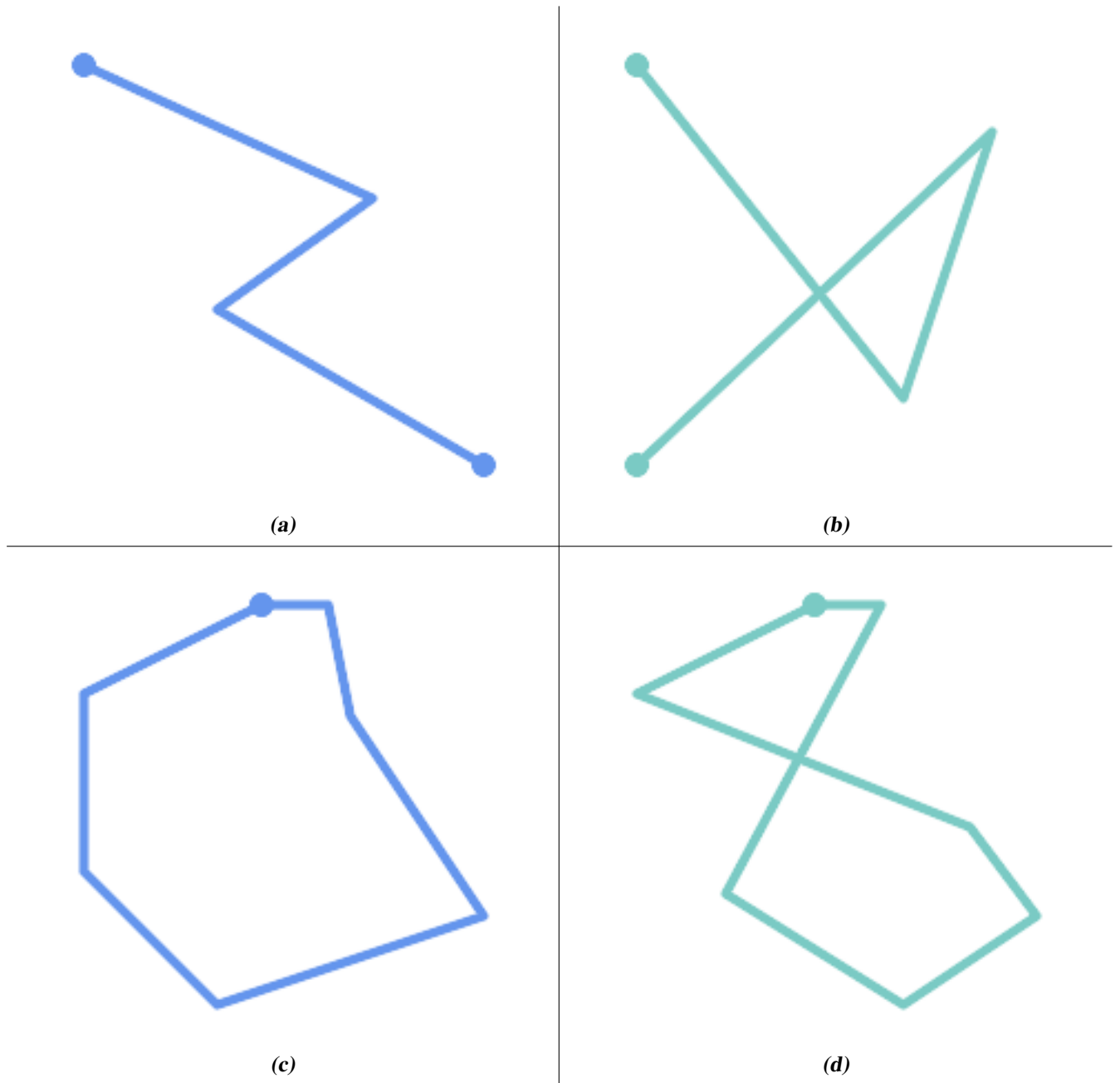
A *simple* geometry is one that has no anomalous geometric points, such as self intersection or self tangency.

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

MULTIPOINTS son simples *simple* si dos coordenadas (POINTS) no son iguales (tienen valores de coordenadas identicos).

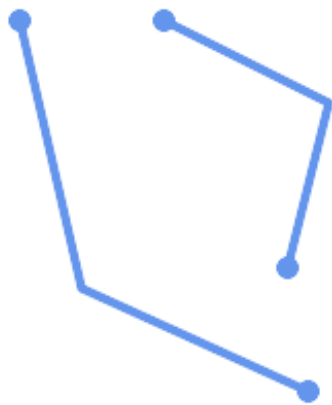
A LINESTRING is *simple* if it does not pass through the same point twice, except for the endpoints. If the endpoints of a simple LineString are identical it is called *closed* and referred to as a Linear Ring.

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

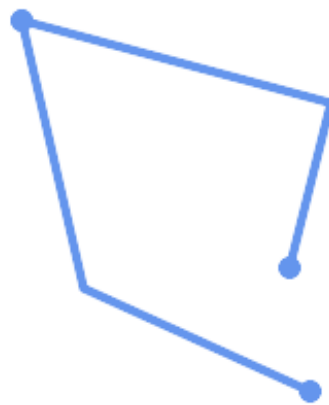


A `MULTILINESTRING` is *simple* only if all of its elements are simple and the only intersection between any two elements occurs at points that are on the boundaries of both elements.

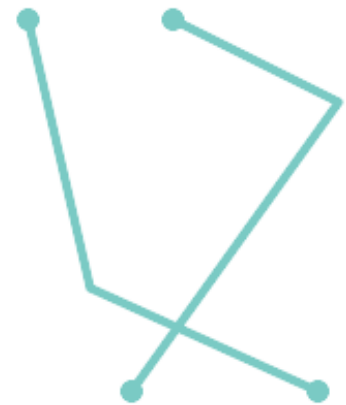
*(e) and (f) are simple MULTILINESTRINGS. (g) is not simple.*



(e)



(f)



(g)

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

To test if a geometry is simple use the **ST\_IsSimple** function:

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

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

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

#### 4.4.2 Valid Geometry

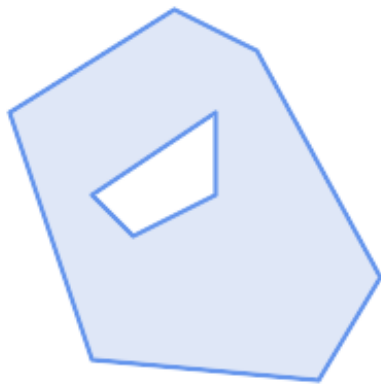
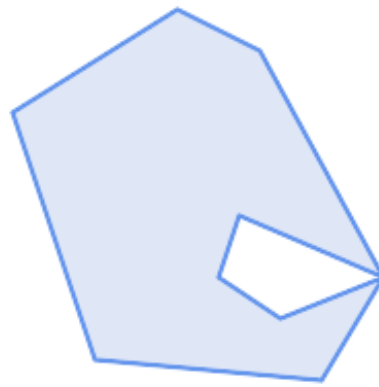
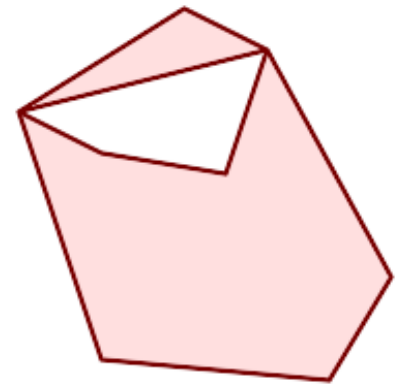
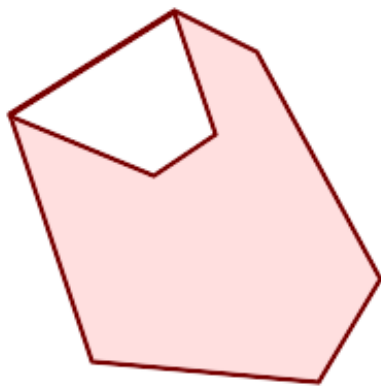
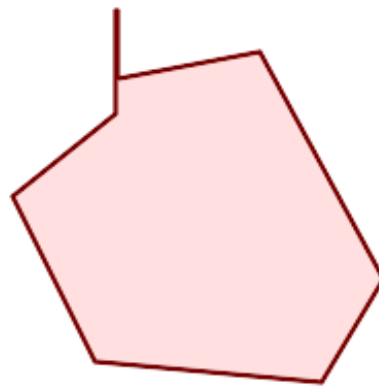
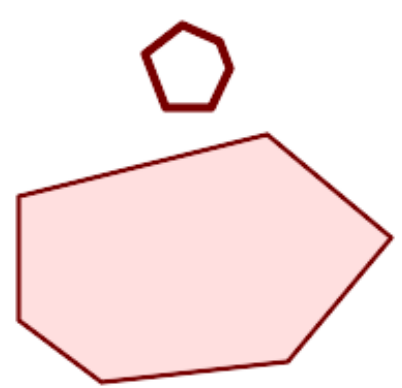
Geometry validity primarily applies to 2-dimensional geometries (POLYGONS and MULTIPOLYGONS). Validity is defined by rules that allow polygonal geometry to model planar areas unambiguously.

A POLYGON is *valid* if:

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

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

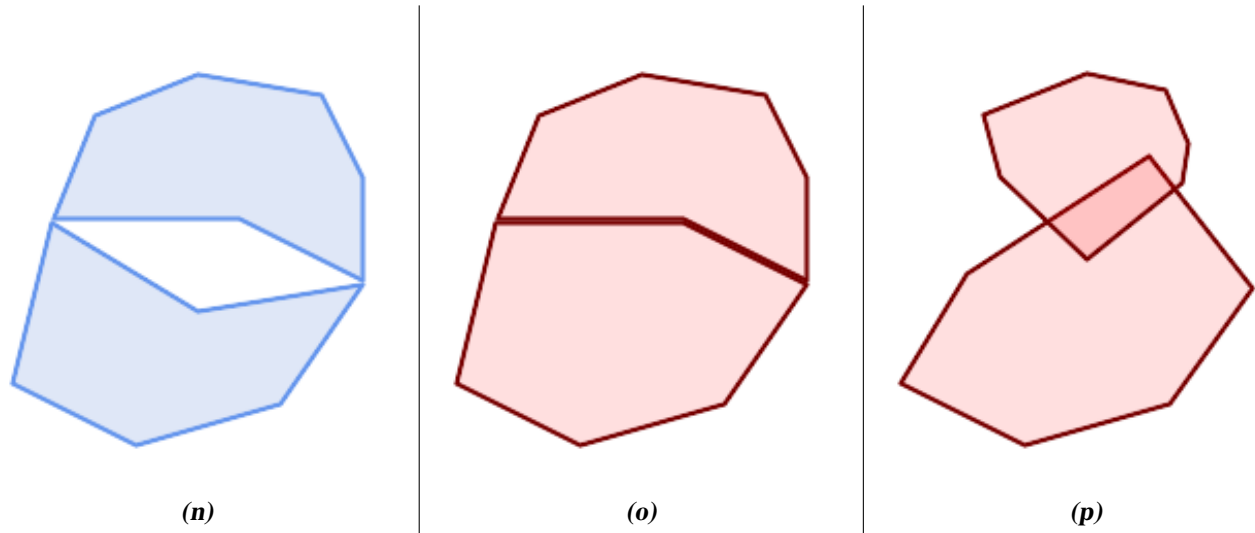


**(h)****(i)****(j)****(k)****(l)****(m)**

A MULTIPOLYGON is *valid* if:

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

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



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

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

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

| len_nonzero | len_zero | self_int |
|-------------|----------|----------|
| t           | f        | t        |

`POINT` and `MULTIPOINT` geometries have no validity rules.

4.4.3 Managing Validity

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

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

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



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

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

```
SELECT ST_IsValid('POLYGON ((20 180, 180 180, 180 20, 20 20, 20 180))');

t
```

Information about the nature and location of an geometry invalidity are provided by the [ST\\_IsValidDetail](#) function:

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

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

In some situations it is desirable to correct invalid geometry automatically. Use the [ST\\_MakeValid](#) function to do this. ([ST\\_MakeValid](#) is a case of a spatial function that *does* allow invalid input!)

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

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

## 4.5 Spatial Reference Systems

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

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

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

A geometry is associated with a Spatial Reference System by its SRID value, which is accessed by [ST\\_SRID](#). The SRID for a geometry can be assigned using [ST\\_SetSRID](#). Some geometry constructor functions allow supplying a SRID (such as [ST\\_Point](#) and [ST\\_MakeEnvelope](#)). The [EWKT](#) format supports SRIDs with the `SRID=n;` prefix.

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

### 4.5.1 SPATIAL\_REF\_SYS Table

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

The `spatial_ref_sys` table definition is:

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

The columns are:

**srid** An integer code that uniquely identifies the [Spatial Reference System](#) (SRS) within the database.

**auth\_name** The name of the standard or standards body that is being cited for this reference system. For example, "EPSG" is a valid `auth_name`.

**auth\_srid** The ID of the Spatial Reference System as defined by the Authority cited in the `auth_name`. In the case of EPSG, this is the EPSG code.

**srtext** La representación Well-Known Text del Sistema de Referencia Espacial (SRS). Un ejemplo de representación WKT SRS es:

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

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

**proj4text** PostGIS uses the PROJ library to provide coordinate transformation capabilities. The `proj4text` column contains the PROJ coordinate definition string for a particular SRID. For example:

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

For more information see the [PROJ web site](#). The `spatial_ref_sys.sql` file contains both `srtext` and `proj4text` definitions for all EPSG projections.

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

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

## 4.5.2 User-Defined Spatial Reference Systems

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

A resource for finding spatial reference systems not defined in the core set is <http://spatialreference.org/>

Some commonly used spatial reference systems are: **4326 - WGS 84 Long Lat**, **4269 - NAD 83 Long Lat**, **3395 - WGS 84 World Mercator**, **2163 - US National Atlas Equal Area**, and the 60 WGS84 UTM zones. UTM zones are one of the most ideal for measurement, but only cover 6-degree regions. (To determine which UTM zone to use for your area of interest, see the **utmzone PostGIS plpgsql helper function**.)

US states use State Plane spatial reference systems (meter or feet based) - usually one or 2 exists per state. Most of the meter-based ones are in the core set, but many of the feet-based ones or ESRI-created ones will need to be copied from [spatialreference.org](http://spatialreference.org).

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

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

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

## 4.6 Spatial Tables

### 4.6.1 Crear una tabla espacial

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

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

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

- the **spatial type modifier** restricts the kind of shapes and dimensions allowed in the column. The value can be any of the supported **geometry subtypes** (e.g. POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION, etc). The modifier supports coordinate dimensionality restrictions by adding suffixes: Z, M and ZM. For example, a modifier of 'LINESTRINGM' allows only linestrings with three dimensions, and treats the third dimension as a measure. Similarly, 'POINTZM' requires four dimensional (XYZM) data.
- the **SRID modifier** restricts the **spatial reference system** SRID to a particular number. If omitted, the SRID defaults to 0.

Examples of creating tables with geometry columns:

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

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

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

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

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

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

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

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

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

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

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

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

## 4.6.2 GEOMETRY\_COLUMNS View

The OGC *Simple Features Specification for SQL* defines the `GEOMETRY_COLUMNS` metadata table to describe geometry table structure. In PostGIS `geometry_columns` is a view reading from database system catalog tables. This ensures that the spatial metadata information is always consistent with the currently defined tables and views. The view structure is:

```
\d geometry_columns
```

View "public.geometry\_columns"

| Column            | Type                   | Modifiers |
|-------------------|------------------------|-----------|
| f_table_catalog   | character varying(256) |           |
| f_table_schema    | character varying(256) |           |
| f_table_name      | character varying(256) |           |
| f_geometry_column | character varying(256) |           |
| coord_dimension   | integer                |           |
| srid              | integer                |           |
| type              | character varying(30)  |           |

The columns are:

**f\_table\_catalog, f\_table\_schema, f\_table\_name** The fully qualified name of the feature table containing the geometry column. There is no PostgreSQL analogue of "catalog" so that column is left blank. For "schema" the PostgreSQL schema name is used (public is the default).

**f\_geometry\_column** El nombre de la columna de geometrías de la tabla de objetos espaciales.

**coord\_dimension** The coordinate dimension (2, 3 or 4) of the column.

**srid** The ID of the spatial reference system used for the coordinate geometry in this table. It is a foreign key reference to the `spatial_ref_sys` table (see Section 4.5.1).

**type** El tipo de objeto espacial. Para restringir la columna espacial a un tipo unico, utiliza uno de: POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION o su version correspondiente de XYM POINTM, LINESTRINGM, POLYGONM, MULTIPOINTM, MULTILINESTRINGM, MULTIPOLYGONM, GEOMETRYCOLLECTIONM. Para colecciones heterogéneas (tipos mixtos), puedes utilizar "GEOMETRY" como tipo.

### 4.6.3 Manually Registering Geometry Columns

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

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

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

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

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

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

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

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

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

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

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

Si ejecutamos en psql

```
\d pois_ny;
```

Vemos que están definidas de forma diferente -- una es typmod, la otra por restricciones.

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

En geometry\_columns, ambas se registran de forma correcta

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

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

De todas formas -- si queremos crear una vista de la siguiente forma

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

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

La columna de la vista basada en typmos se registra de forma correcta, pero la basada en restricciones no.

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

This may change in future versions of PostGIS, but for now to force the constraint-based view column to register correctly, you need to do this:

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



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

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

## 4.7 Loading Spatial Data

Once you have created a spatial table, you are ready to upload spatial data to the database. There are two built-in ways to get spatial data into a PostGIS/PostgreSQL database: using formatted SQL statements or using the Shapefile loader.

### 4.7.1 Using SQL to Load Data

If spatial data can be converted to a text representation (as either WKT or WKB), then using SQL might be the easiest way to get data into PostGIS. Data can be bulk-loaded into PostGIS/PostgreSQL by loading a text file of SQL `INSERT` statements using the `psql` SQL utility.

A SQL load file (`roads.sql` for example) might look like this:

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

The SQL file can be loaded into PostgreSQL using `psql`:

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

### 4.7.2 Using the Shapefile Loader

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

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

**(claldp) Estas opciones son exclusivas entre ellas:**

- c Creates a new table and populates it from the Shapefile. *This is the default mode.*
- a Appends data from the Shapefile into the database table. Note that to use this option to load multiple files, the files must have the same attributes and same data types.
- d Drops the database table before creating a new table with the data in the Shapefile.

- p** Solo produce el código del comando SQL de creación de la tabla, sin añadir ningún dato. Esto puede utilizarse si necesitas separar completamente los pasos de creación de la tabla y de carga de datos
- ?** Muestra la ayuda en pantalla.
- D** Utiliza el formato "dump" de PostgreSQL en la salida de datos. Esto puede combinarse con -a, -c, y -d. Es mucho mas rápido cargar este fichero "dump" que utilizando en comando SQL "INSERT" por defecto. Utiliza esto ara grandes conjuntos de datos.
- s** [**<FROM\_SRID>**]:**<SRID>** Creates and populates the geometry tables with the specified SRID. Optionally specifies that the input shapefile uses the given FROM\_SRID, in which case the geometries will be reprojected to the target SRID.
- k** Mantiene las mayúsculas en los identificadores (columnas, esquemas y atributos). Observa que los atributos en los shapefiles están siempre en MAYÚSCULAS.
- i** Fuerza la creación de enteros a enteros estándar de 32-bits, no crea enteros bigint de 64-bits, aunque la firma de la cabecera del DBF parezca que lo garantiza.
- I** Crea un índice GiST de la columna de geometrías.
- m** -m *a\_file\_name* Especifica un fichero que contiene un conjunto de asignaciones de nombres (largos) de columnas a nombres de columna DBF de 10 caracteres. El contenido del archivo es una o más líneas de dos nombres separados por espacios en blanco y no se arrastra o espacio inicial. Por ejemplo:
 

```
COLUMNNAME DBFFIELD1
AVERYLONGCOLUMNNAME DBFFIELD2
```
- S** Genera geometrías simples en lugar de MULTI geometrías. Solo funcionará si todas las geometrías son actualmente simples (I.E. un MULTIPOLYGON con una única capa, o un MULTIPOINT con un único vértice).
- t** **<dimensionality>** Fuerza a que la geometría de salida tenga la dimensión especificada. Utiliza las siguientes cadenas para indicar la dimensión: 2D, 3DZ, 3DM, 4D.  
Si la entrada tiene menos dimensiones de las especificadas, la salida tendrá estas dimensiones rellenas con ceros. Si la entrada tiene mas dimensiones de las especificadas, las dimensiones no deseadas se eliminarán.
- w** Salida en formato WKT, en vez de WKB. Observa que esto puede introducir derivas en las coordenadas debido a la perdida de precisión.
- e** Ejecuta cada sentencia una por una, sin utilizar una transacción. Esto permite cargar la mayoría de datos correctos cuando existen algunas geometrías no validas que generan errores. Observa que esta opción no se puede utilizar con -D ya que el formato "dump" siempre utiliza transacciones.
- W** **<encoding>** Especifica la codificación de los datos de entrada (fichero dbf). Cuando se utiliza, todos los atributos del fichero dbf son convertidos desde la codificación especificada a UTF8. La salida SQL resultante contendrá un comando SET CLIENT\_ENCODING to UTF8, así que el backend sera capaz de reconvertir desde UTF8 a cualquier codificación que este configurada en la base de datos para uso interno.
- N** **<policy>** Políticas de gestión de geometrías NULL (insert\*, skip, abort)
- n** -n solo importa los ficheros dbf. Si tus datos no tienen shapefiles correspondientes, se cambiara de forma automática a este modo y se cargara únicamente el dbf. Así que esta opción solo se necesita si lo unifico que quieres cargar son los atributos y no las geometrías.
- G** Utiliza el tipo "geography" en lugar del tipo "geometry" (requiere datos en lon/lat) en WGS84 long lat (SRID=4326)
- T** **<tablespace>** Especifica el "tablespace" para la nueva tabla. Los índices seguirán utilizando el "tablespace" por defecto a menos que el parámetro -X este en uso. La documentación de PostgreSQL tiene una buena descripción de los "tablespaces" personalizados.
- X** **<tablespace>** Especifica el "tablespace" para los índices de la nueva tabla. Esto se aplica a los índices de clave primaria y a los índices espaciales GiST si se usa también la opción -I.

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

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

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

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

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

## 4.8 Extracting Spatial Data

Spatial data can be extracted from the database using either SQL or the Shapefile dumper. The section on SQL presents some of the functions available to do comparisons and queries on spatial tables.

### 4.8.1 Using SQL to Extract Data

The most straightforward way of extracting spatial data out of the database is to use a SQL SELECT query to define the data set to be extracted and dump the resulting columns into a parsable text file:

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

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

There will be times when some kind of restriction is necessary to cut down the number of records returned. In the case of attribute-based restrictions, use the same SQL syntax as used with a non-spatial table. In the case of spatial restrictions, the following functions are useful:

**ST\_Intersects** This function tells whether two geometries share any space.

= Este test comprueba si dos geometrías son geoméricamente idénticas. Por ejemplo, si 'POLYGON((0 0,1 1,1 0,0 0))' es la misma que 'POLYGON((0 0,1 1,1 0,0 0))' (si que lo es).

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

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

La consulta anterior deberá devolver el único registro de la tabla "ROADS\_GEOM" cuya geometría era igual a este valor.

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

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

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

Cuando utilizamos el operador "&&", puedes especificar ya sea un BOX3D como la función de comparación o una GEOMETRY. Cuando se especifica una geometría, sin embargo, se utiliza para la comparación su cuadro delimitador (bounding box).

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

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

Observa el uso del SRID 123, para especificar la proyección de la envolvente.

## 4.8.2 Using the Shapefile Dumper

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

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

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

Las opciones del comando son:

- f <filename>** Escribe la salida en un fichero con un nombre particular
- h <host>** Especifica el servidor al que conectarse.
- p <port>** Especifica el puerto del servidor de la base de datos al que conectarse.
- P <password>** La contraseña a utilizar en la conexión de la base de datos.
- u <user>** El nombre del usuario a utilizar en la conexión a la base de datos.
- g <geometry column>** En el caso que las tablas tengan varias columnas de geometrías, la columna de geometrías a utilizar cuando se escriba el fichero shape.
- b** Utiliza un cursor binario. Esto hace las operaciones mas rápido, pero no funcionará si algún atributo NO-geométrico de la tabla carece de conversión a texto.
- r** Modo Raw. No suprime el campo `gid`, o omite los nombres de las columnas.
- m filename** Reasignar los identificadores de diez nombres de los personajes. El contenido del archivo son líneas de dos símbolos separados por un único espacio en blanco y sin espacios al final, o al inicio: `VERYLONGSYMBOL SHORTONE ANOTHERVERYLONGSYMBOL SHORTER` etc.

## 4.9 Spatial Indexes

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

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

- **GiST (Generalized Search Tree)** indexes break up data into "things to one side", "things which overlap", "things which are inside" and can be used on a wide range of data-types, including GIS data. PostGIS uses an R-Tree index implemented on top of GiST to index spatial data. GiST is the most commonly-used and versatile spatial index method, and offers very good query performance.
- **BRIN (Block Range Index)** indexes operate by summarizing the spatial extent of ranges of table records. Search is done via a scan of the ranges. BRIN is only appropriate for use for some kinds of data (spatially sorted, with infrequent or no update). But it provides much faster index create time, and much smaller index size.
- **SP-GiST (Space-Partitioned Generalized Search Tree)** is a generic index method that supports partitioned search trees such as quad-trees, k-d trees, and radix trees (tries).

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

See also the [PostGIS Workshop section on spatial indexes](#), and the [PostgreSQL manual](#).

#### 4.9.1 Indices GiST

GiST stands for "Generalized Search Tree" and is a generic form of indexing for multi-dimensional data. PostGIS uses an R-Tree index implemented on top of GiST to index spatial data. GiST is the most commonly-used and versatile spatial index method, and offers very good query performance. Other implementations of GiST are used to speed up searches on all kinds of irregular data structures (integer arrays, spectral data, etc) which are not amenable to normal B-Tree indexing. For more information see the [PostgreSQL manual](#).

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

La sintaxis para la creación de un índice GiST en una columna "geometry" es como sigue:

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

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

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

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

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

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

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

#### 4.9.2 BRIN Indexes

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

Once a spatial table exceeds a few thousand rows, you will want to build an index to speed up spatial searches of the data. GiST indexes are very performant as long as their size doesn't exceed the amount of RAM available for the database, and as long as you can afford the index storage size, and the cost of index update on write. Otherwise, for very large tables BRIN index can be considered as an alternative.

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

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

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

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

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

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

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

The syntax for building a BRIN index on a geometry column is:

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

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

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

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

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

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

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

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

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

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

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

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

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

To summarize using BRIN for spatial data:

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

### 4.9.3 SP-GiST Indexes

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

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

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

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

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

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

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

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

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

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

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

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

There is no support for kNN searches at the moment.

#### 4.9.4 Tuning Index Usage

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

If you find your spatial indexes are not being used, there are a few things you can do:

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

You should regularly vacuum your databases anyways. Many PostgreSQL DBAs run `VACUUM` as an off-peak cron job on a regular basis.

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

For more information see the Postgres manual section on [Query Planning](#).



## Chapter 5

# Consulta Espacial

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

### 5.1 Determining Spatial Relationships

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

#### 5.1.1 Dimensionally Extended 9-Intersection Model

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

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

##### Boundary

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

##### Interior

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

##### Exterior

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

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

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

- 0 => point

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

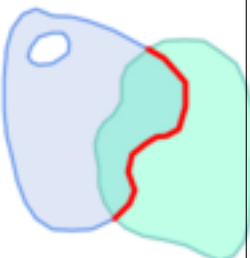
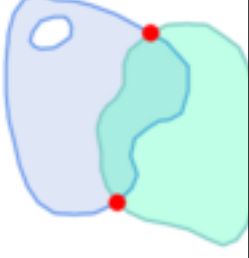
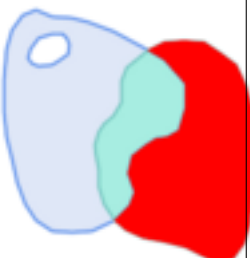
Using this notation, the intersection matrix for two geometries *a* and *b* is:

|          | Interior                 | Boundary                 | Exterior                 |
|----------|--------------------------|--------------------------|--------------------------|
| Interior | $\dim( I(a) \cap I(b) )$ | $\dim( I(a) \cap B(b) )$ | $\dim( I(a) \cap E(b) )$ |
| Boundary | $\dim( B(a) \cap I(b) )$ | $\dim( B(a) \cap B(b) )$ | $\dim( B(a) \cap E(b) )$ |
| Exterior | $\dim( E(a) \cap I(b) )$ | $\dim( E(a) \cap B(b) )$ | $\dim( E(a) \cap E(b) )$ |

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





|          | Interior                                                                                                            | Boundary                                                                                                             | Exterior                                                                                                              |
|----------|---------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| Interior | <br>$\dim( I(a) \cap I(b) ) = 2$   | <br>$\dim( I(a) \cap B(b) ) = 1$   | <br>$\dim( I(a) \cap E(b) ) = 2$   |
| Boundary | <br>$\dim( B(a) \cap I(b) ) = 1$  | <br>$\dim( B(a) \cap B(b) ) = 0$  | <br>$\dim( B(a) \cap E(b) ) = 1$  |
| Exterior | <br>$\dim( E(a) \cap I(b) ) = 2$ | <br>$\dim( E(a) \cap B(b) ) = 1$ | <br>$\dim( E(a) \cap E(b) ) = 2$ |

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

For more information, refer to:

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

5.1.2 Named Spatial Relationships

To make it easy to determine common spatial relationships, the OGC SFS defines a set of *named spatial relationship predicates*. PostGIS provides these as the functions **ST\_Contains**, **ST\_Crosses**, **ST\_Disjoint**, **ST\_Equals**, **ST\_Intersects**, **ST\_Overlaps**, **ST\_Touches**, **ST\_Within**. It also defines the non-standard relationship predicates **ST\_Covers**, **ST\_CoveredBy**, and **ST\_ContainsProperly**.

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

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

For more details and illustrations, see the [PostGIS Workshop](#).

### 5.1.3 General Spatial Relationships

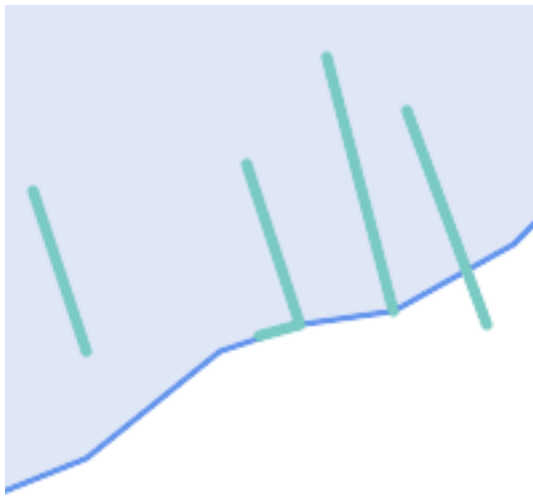
In some cases the named spatial relationships are insufficient to provide a desired spatial filter condition.



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

A two-step solution would be to first compute the actual intersection (`ST_Intersection`) of pairs of road lines that spatially intersect (`ST_Intersects`), and then check if the intersection's `ST_GeometryType` is 'LINESTRING' (properly dealing with cases that return `GEOMETRYCOLLECTIONS` of `[MULTI] POINTs`, `[MULTI] LINESTRINGs`, etc.).

Clearly, a simpler and faster solution is desirable.



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

- `ST_Contains(lake, wharf) = TRUE`
  - `ST_ContainsProperly(lake, wharf) = FALSE`
  - `ST_GeometryType(ST_Intersection(wharf, lake)) = 'LINESTRING'`
  - `ST_NumGeometries(ST_Multi(ST_Intersection(ST_Boundary(wharf), ST_Boundary(lake)))) = 1`
- ... but needless to say, this is quite complicated.

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

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

1010F0212
```

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

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

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

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

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

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

## 5.2 Using Spatial Indexes

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

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

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

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

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

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

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

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

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

For more information and examples see the [PostGIS Workshop](#).

## 5.3 Examples of Spatial SQL

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

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

The definition of the `bc_municipality` table is:

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

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

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

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

km_roads

70842.1243039643
```

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

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

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

hectares

32657.9103824927
```

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

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

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

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

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

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

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

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

```

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

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

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

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

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

```

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

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

```

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

kilometers

4.89151904172838
```

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

```

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

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



## Chapter 6

# Consejos de rendimiento

### 6.1 Tablas pequeñas de geometrías grandes

#### 6.1.1 Descripción del problema

Versiones actuales de PostgreSQL (incluyendo la 8.0) tienen algunas debilidades en la optimización de consultas respecto a tablas TOAST. Las tablas TOAST son una especie de "cámara de extensiones" utilizadas para almacenar valores grandes (en sentido de tamaño de datos) que no se pueden mostrar en páginas de datos (como textos largos, imágenes o geometrías complejas con muchos vértices). Para mas información visita [the PostgreSQL Documentation for TOAST](#)

El problema aparece si ocurre que tienes una tabla con geometrías bastante grandes, pero no demasiadas filas de ellas (como una tabla que contiene los límites de todos los países europeos en alta resolución). A continuación, la tabla en sí es pequeña, pero utiliza una gran cantidad de espacio TOAST. En nuestro caso de ejemplo, la tabla en sí tenía alrededor de 80 filas y se utiliza sólo 3 páginas de datos, pero la tabla TOAST utiliza 8225 páginas.

Ahora al emitir una consulta en la que utilizas el operador geométrico `&&` para buscar un límite que coincide sólo unas pocas de esas filas, el optimizador de consultas ve que la tabla sólo tiene 3 páginas y 80 filas. Se estima que un escaneo secuencial en una tabla pequeña de este tipo es mucho más rápida que usando un índice. Y por lo que decide ignorar el índice de GIST. Por lo general, esta estimación es correcta. Pero en nuestro caso, el operador `&&` tiene que buscar en cada geometría del disco la comparación de los límites, y leer todas las páginas TOAST también.

Para comprobar si padeces de este error, utiliza el comando "EXPLAIN ANALYZE" postgresql. Para obtener más información y los detalles técnicos, puedes leer el hilo en la lista de correo de rendimiento postgres: <http://archives.postgresql.org/pgsql-performance/2005-02/msg00030.php>

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

#### 6.1.2 Soluciones provisionales

La gente de PostgreSQL esta intentando resolver este problema haciendo la estimación de la consulta compatible con TOAST. Por el momento, aquí van dos soluciones provisionales:

La primera consiste en forzar la consulta a utilizar índices. Envía "SET enable\_seqscan TO off;" al servidor antes de ejecutar la consulta. Esto, básicamente fuerza al planificador de consultas a evitar exploraciones secuenciales siempre que sea posible. Por lo tanto, utiliza el índice GIST como de costumbre. Pero este comando debe ser establecido en cada conexión, y hace que el planeador de consultas cometa errores de estimación en otros casos, por lo que debes enviar al servidor "SET enable\_seqscan TO on;" después de la consulta.

La segunda solución es hacer el escaneo secuencia tan rápido como el planificador de consultas cree. Esto, se puede lograr creando una consulta que "cachee" los límites o bbox, y hacer coincidir en contra de esta. En nuestro ejemplo, los comandos son:

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

Ahora cambia tu consulta para utilizar el operador espacial `&&` con `bbox` en vez de `geom_column`, así:

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

Por supuesto, si añades o cambias filas de "mutable", tienes que mantener el campo `bbox` sincronizado. La forma mas transparente de hacerlo son los triggers o funciones disparadoras, pero también puedes modificar tu aplicación para mantener la columna `bbox` o ejecutar la consulta `UPDATE` siguiente después de cada modificación.

## 6.2 CLUSTERing o indices geométricos

Para las tablas que en su mayoría son de sólo lectura, y donde se utiliza un índice único para la mayoría de las consultas, PostgreSQL ofrece el comando `CLUSTER`. Este comando reordena físicamente todas las filas de datos en el mismo orden que los criterios de índice, dando dos ventajas de rendimiento: En primer lugar, para los recorridos de intervalo del índice, el número de búsquedas en la tabla de datos se reduce drásticamente. En segundo lugar, si el conjunto de trabajo se concentra en algunos intervalos pequeños en los índices, tienes un caché más eficiente porque las filas de datos se distribuyen a lo largo de un menor número de páginas de datos. (Te invitamos a leer la documentación de comandos `CLUSTER` del manual de PostgreSQL sobre este tema.)

De todas formas, PostgreSQL no permite el "clustering" en índices GiST de PostGIS por que los índices GiST simplemente ignoran los valores `NULL`, tendrás el siguiente mensaje de error:

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

Como sugiere el mensaje de ayuda, podemos evitar esta deficiencia añadiendo una restricción "not null" a la tabla:

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

Por supuesto, esto no funcionará si necesitas valores `NULL` en tu columna de geometrías. Adicionalmente, debes utilizar el método anterior para añadir la restricción, utilizando restricciones `CHECK` como "ALTER TABLE blubb ADD CHECK (geómetra is not nulo);" no funcionara.

## 6.3 Evitar la conversión de dimensión

A veces, sucede que tienes datos en 3D o 4D en tus tablas pero siempre, al acceder a ella utilizando funciones conformes con OpenGIS como `ST_AsText()` o `ST_AsBinary()`, sólo devuelven geometrías 2D de salida. Esto ocurre por que lo hacen llamando internamente a la función `ST_Force_2d()`, que introduce una sobrecarga significativa para geometrías grandes. Para evitar esta sobrecarga, puede ser factible comprobar la validez de suprimir esas dimensiones adicionales de una vez por todas:

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

Ten en cuenta que si las has añadido a tu columna de geometría utilizando `addGeometryColumn()` habrá una restricción en la dimensión de la geometría. Para pasar la restricción por alto tendrás que quitarla. Recuerda actualizar la entrada en la tabla `geometry_columns` y volver a crear la restricción después.

En el caso de tablas de gran tamaño, puede ser conveniente dividir este `UPDATE` en porciones más pequeñas, restringiendo la actualización de una parte de la tabla a través de una cláusula `WHERE` y su clave primaria o de otros criterios, y la ejecución de un simple "VACUUM"; entre los `UPDATE`. Esto reduce drásticamente la necesidad de espacio de disco temporal. Además, si has mezclado dimensiones de geometrías, que restringen el `UPDATE` con "WHERE dimension(the\_geom)>2" salta la reescritura de geometrías que ya están en 2D.

# Chapter 7

## Manual de Referencia PostGIS

Las siguientes funciones son las que probablemente necesite un usuario PostGIS . Existen otras funciones de soporte necesarias para los objetos PostGIS que no se usan por la mayoría de usuarios.



**Note**  
PostGIS ha comenzado una transición de la convención de nomenclatura existente, a una convención SQL-MM-céntrica. Como resultado, la mayoría de las funciones que conoces y adoras han sido renombradas usando el prefijo espacial estándar (ST). Funciones anteriores están todavía disponibles, aunque no se enumeran en este documento donde las funciones actualizadas son equivalentes. Las funciones no st\_ no mencionadas en esta documentación están en desuso y se eliminarán en una versión futura de modo que DEJA DE UTILIZARLAS.

### 7.1 Tipos Geometry/Geography/Box en PostgreSQL PostGIS

#### 7.1.1 box2d

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

##### Descripción

box3d es un tipo de dato espacial usado para representar la caja que contiene una geometría o un grupo de geometrías. ST\_3DExtent devuelve un objeto box3d.

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

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

##### Comportamiento de la conversión de tipo de dato

Esta sección detalla los cambios de tipo automáticos y explícitos permitidos para este tipo de dato

| Convertir a | Comportamiento |
|-------------|----------------|
| box3d       | automatic      |
| geometry    | automatic      |

**Vea también**

Section [12.7](#)

**7.1.2 box3d**

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

**Descripción**

box3d es un tipo de dato espacial usado para representar la caja que contiene una geometría o un grupo de geometrías. ST\_3DExtent devuelve un objeto box3d.

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

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

**Comportamiento de la conversión de tipo de dato**

Esta sección detalla los cambios de tipo automáticos y explícitos permitidos para este tipo de dato

| Convertir a | Comportamiento |
|-------------|----------------|
| box         | automatic      |
| box2d       | automatic      |
| geometry    | automatic      |

**Vea también**

Section [12.7](#)

**7.1.3 geometry**

geometry — geography es un tipo de dato espacial usado para representar una feature en un sistema de coordenadas de Tierra esférica.

**Descripción**

geometry es un tipo de datos postgis fundamental, usado para representar una feature en un sistema de coordenadas euclidiano.

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

**Comportamiento de la conversión de tipo de dato**

Esta sección detalla los cambios de tipo automáticos y explícitos permitidos para este tipo de dato

| Convertir a | Comportamiento |
|-------------|----------------|
| box         | automatic      |
| box2d       | automatic      |
| box3d       | automatic      |
| bytea       | automatic      |
| geography   | automatic      |
| text        | automatic      |

**Vea también**

Section 4.1, Section 4.3

### 7.1.4 geometry\_dump

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

**Descripción**

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

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

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

**Vea también**

Section 12.6

### 7.1.5 geography

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

**Descripción**

`geography` es un tipo de dato espacial usado para representar una feature en un sistema de coordenadas de Tierra esférica.

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

**Comportamiento de la conversión de tipo de dato**

Esta sección detalla los cambios de tipo automáticos y explícitos permitidos para este tipo de dato

| Convertir a           | Comportamiento        |
|-----------------------|-----------------------|
| <code>geometry</code> | <code>explicit</code> |

**Vea también**

Section 4.3, Section 4.3

## 7.2 Funciones de Gestión

### 7.2.1 AddGeometryColumn

`AddGeometryColumn` — Suprime una columna de geometrías de una tabla espacial.

---

## Synopsis

```
text AddGeometryColumn(varchar table_name, varchar column_name, integer srid, varchar type, integer dimension, boolean use_tpmmod=true);
```

```
text AddGeometryColumn(varchar schema_name, varchar table_name, varchar column_name, integer srid, varchar type, integer dimension, boolean use_tpmmod=true);
```

```
text AddGeometryColumn(varchar catalog_name, varchar schema_name, varchar table_name, varchar column_name, integer srid, varchar type, integer dimension, boolean use_tpmmod=true);
```

## Descripción

Añade una columna de geometría a una tabla existente de atributos. `schema_name` es el nombre del esquema de la tabla. `srid` debe ser una referencia de valor entero a una entrada en la tabla `SPATIAL_REF_SYS`. `type` debe ser una cadena que corresponde al tipo de geometría, por ejemplo, 'POLYGON' or 'MULTILINESTRING'. Se lanza un error si no existe el `schemaname` (o no esta visible en el `search_path` actual) o el `SRID`, el tipo de geometría, o la dimensión no son validos.

### Note



Cambiado: 2.0.0 Esta función ya no se actualiza desde `geometry_columns` ya que `geometry_columns` es una vista que se lee desde los catálogos del sistema. Por defecto tampoco crea las restricciones, sino que utiliza el modificador de tipo de PostgreSQL. Así que para la construcción de una columna de tipo POINT en wgs84 con esta función ejemplo que hoy es equivalente a: `ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326);`  
Cambiado: 2.0.0 Si necesitas el comportamiento antiguo de restricciones, utiliza el valor predeterminado `use_tpmmod`, pero cambiala a `false`.

### Note



Cambiado: 2.0.0 Las Vistas ya no pueden ser registradas manualmente en `geometry_columns`, no obstante las vistas se que construyan a partir de geometrías `tpmmod` de las tablas de geometrías y sean utilizadas sin funciones wrapper se registrarán correctamente porque heredan el comportamiento `tpmmod` de su columna de la tabla padre. Las vistas que utilizan funciones de geometría que devuelvan geometrías necesitarán de transformación `cast` a geometrías `tpmmod` para esta columnas de geometrías de la vista y que se registren correctamente en `geometry_columns`. Consulta Section 4.6.3.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



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



This method supports Circular Strings and Curves.

Mejorada: 2.0.0 introducción del argumento `use_tpmmod`. El valor predeterminado es crear columnas de geometrías basadas en `tpmmod` en lugar de las basadas en restricciones.

## Ejemplos

```
-- Create schema to hold data
CREATE SCHEMA my_schema;
-- Create a new simple PostgreSQL table
CREATE TABLE my_schema.my_spatial_table (id serial);
```

```
-- Describing the table shows a simple table with a single "id" column.
postgis=# \d my_schema.my_spatial_table
```

| Column | Type   | Modifiers |
|--------|--------|-----------|
| id     | serial |           |

```
id | integer | not null default nextval('my_schema.my_spatial_table_id_seq'::regclass)

-- Add a spatial column to the table
SELECT AddGeometryColumn ('my_schema','my_spatial_table','geom',4326,'POINT',2);

-- Add a point using the old constraint based behavior
SELECT AddGeometryColumn ('my_schema','my_spatial_table','geom_c',4326,'POINT',2, false);

--Add a curvepolygon using old constraint behavior
SELECT AddGeometryColumn ('my_schema','my_spatial_table','geomcp_c',4326,'CURVEPOLYGON',2, ←
 false);

-- Describe the table again reveals the addition of a new geometry columns.
\d my_schema.my_spatial_table
 addgeometrycolumn

my_schema.my_spatial_table.geomcp_c SRID:4326 TYPE:CURVEPOLYGON DIMS:2
(1 row)
```

| Table "my_schema.my_spatial_table" |                      |  |                                                                               |
|------------------------------------|----------------------|--|-------------------------------------------------------------------------------|
| Column                             | Type                 |  | Modifiers                                                                     |
| id                                 | integer              |  | not null default nextval('my_schema. ←<br>my_spatial_table_id_seq'::regclass) |
| geom                               | geometry(Point,4326) |  |                                                                               |
| geom_c                             | geometry             |  |                                                                               |
| geomcp_c                           | geometry             |  |                                                                               |

Check constraints:

```
"enforce_dims_geom_c" CHECK (st_ndims(geom_c) = 2)
"enforce_dims_geomcp_c" CHECK (st_ndims(geomcp_c) = 2)
"enforce_geotype_geom_c" CHECK (geometrytype(geom_c) = 'POINT'::text OR geom_c IS NULL)
"enforce_geotype_geomcp_c" CHECK (geometrytype(geomcp_c) = 'CURVEPOLYGON'::text OR ←
 geomcp_c IS NULL)
"enforce_srid_geom_c" CHECK (st_srid(geom_c) = 4326)
"enforce_srid_geomcp_c" CHECK (st_srid(geomcp_c) = 4326)
```

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

```
SELECT f_geometry_column As col_name, type, srid, coord_dimension As ndims
FROM geometry_columns
WHERE f_table_name = 'my_spatial_table' AND f_table_schema = 'my_schema';
```

| col_name | type         | srid | ndims |
|----------|--------------|------|-------|
| geom     | Point        | 4326 | 2     |
| geom_c   | Point        | 4326 | 2     |
| geomcp_c | CurvePolygon | 4326 | 2     |

También puedes ver

[DropGeometryColumn](#), [DropGeometryTable](#), [Section 4.6.2](#), [Section 4.6.3](#)

### 7.2.2 DropGeometryColumn

DropGeometryColumn — Suprime una columna de geometrías de una tabla espacial.

## Synopsis

```
text DropGeometryColumn(varchar table_name, varchar column_name);
text DropGeometryColumn(varchar schema_name, varchar table_name, varchar column_name);
text DropGeometryColumn(varchar catalog_name, varchar schema_name, varchar table_name, varchar column_name);
```

## Descripción

Suprime una columna de geometrías de una tabla espacial. Observa que `schema_name` debe apuntar al campo `f_table_schema` del registro de la tabla `geometry_columns`.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



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



This method supports Circular Strings and Curves.



### Note

Cambiado: 2.0.0 Se proporciona esta función para la compatibilidad con versiones anteriores. Ahora que `geometry_columns` es una vista y no un catálogo del sistema, se puede eliminar una columna de geometría como cualquier otra columna de la tabla utilizando `ALTER TABLE`

## Ejemplos

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

my_schema.my_spatial_table.geom effectively removed.

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

## También puedes ver

[AddGeometryColumn](#), [DropGeometryTable](#), [Section 4.6.2](#)

## 7.2.3 DropGeometryTable

`DropGeometryTable` — Borra una tabla y todas sus referencias en la tabla `geometra_columns`.

## Synopsis

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



## Descripción

Borra la tabla y todas sus referencias en la tabla `geometra_column`. Nota: utiliza el esquema `current_schema()` de una instalación `pgsql` si el esquema no se especifica.



### Note

Cambiado: 2.0.0 Se proporciona esta función para la compatibilidad con versiones anteriores. Ahora que `geometry_columns` es una vista y no un catálogo del sistema, se puede borrar una tabla con columnas de geometría como cualquier otra tabla utilizando `DROP TABLE`

## Ejemplos

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

-- The above is now equivalent to --
DROP TABLE my_schema.my_spatial_table;
```

## También puedes ver

[AddGeometryColumn](#), [DropGeometryColumn](#), [Section 4.6.2](#)

## 7.2.4 Find\_SRID

`Find_SRID` — Returns the SRID defined for a geometry column.

## Synopsis

integer **Find\_SRID**(varchar a\_schema\_name, varchar a\_table\_name, varchar a\_geomfield\_name);

## Descripción

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

## Ejemplos

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

4269
```

## También puedes ver

[ST\\_SRID](#)

## 7.2.5 Populate\_Geometry\_Columns

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

## Synopsis

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

## Descripción

Asegura que las columnas de geometría se define con modificadores de tipo o tienen restricciones espaciales apropiadas. Esto asegura que se registrarán correctamente en la vista `geometry_columns`. Por defecto se convertirán todas las columnas de geometría sin modificador de tipo a modificadores de tipo. Para conseguir el comportamiento del sistema antiguo selecciona `use_typmod = false`

Para la compatibilidad con versiones anteriores y para necesidades espaciales, como la herencia de tablas, donde cada tabla secundaria puede tener un tipo de geometría diferente, el comportamiento de restricción de comprobación anterior sigue siendo compatible. Si necesita el comportamiento antiguo, debe pasar el nuevo argumento opcional como falso `use_typmod=false`. Cuando se haga esto, las columnas de geometría se crearán sin modificadores de tipo pero tendrán 3 restricciones definidas. En particular, esto significa que cada columna geométrica que pertenezca a una tabla tiene al menos tres restricciones:

- `enforce_dims_the_geom` - asegura que cada geometría posee la misma dimensión (mira en [ST\\_NDims](#))
- `enforce_geotype_the_geom` - asegura que cada geometría es del mismo tipo (mira en [GeometryType](#))
- `enforce_srid_the_geom` - asegura que cada geometría tiene la misma proyección (mira en [ST\\_SRID](#))

Si se da una tabla `oid`, esta función trata de determinar el `srid`, la dimensión, y el tipo de geometría de todas las columnas de geometrías en la tabla, añadiendo las restricciones si es necesario. Si no hay errores, una fila apropiada se insertará en la tabla `geometry_columns`, si hay errores, se captura la excepción y se envía un mensaje de error con la descripción del problema.

Si se da una vista `oid`, como en el caso de una tabla `oid`, esta función trata de determinar el `srid`, la dimensión, y el tipo de geometría de todas las columnas de geometrías en la tabla, añadiendo las filas apropiadas tabla `geometry_columns`, pero no se ejecuta nada para hacer cumplir las restricciones.

La variante sin parámetros es un simple wrapper de la variante con parámetros que trunca y rellena la tabla `geometry_columns` para cada tabla y vista espacial de la base de datos, añadiendo restricciones espaciales apropiadas a cada tabla. Devuelve un sumario de los numero de columnas de geometrías detectadas en la base de datos y el numeroque se insertaron en la tabla `geometry_columns`. La versión con parámetros simplemente devuelve el numero de filas insertado en la tabla `geometry_columns`.

Disponibilidad: 1.4.0

Cambiado: 2.0.0 Por defecto, ahora utiliza modificadores de tipo en lugar de restricciones de tipo check para limitar los tipos de geometría. Puedes seguir utilizando el comportamiento de las restricciones check con el uso de la nueva variable `use_typmod` y estableciéndolo a `false`.

Mejorado: 2.0.0 el argumento opcional `use_typmod` fue introducido y permite controlar si las columnas se crean con modificadores de tipo o con restricciones de tipo check.

## Ejemplos

```
CREATE TABLE public.myspatial_table(gid serial, geom geometry);
INSERT INTO myspatial_table(geom) VALUES(ST_GeomFromText('LINESTRING(1 2, 3 4)',4326));
-- This will now use typ modifiers. For this to work, there must exist data
SELECT Populate_Geometry_Columns('public.myspatial_table'::regclass);

populate_geometry_columns

1

\d myspatial_table
```

```
Table "public.myspatial_table"
Column | Type | Modifiers
-----+-----+-----
gid | integer | not null default nextval('myspatial_table_gid_seq'::regclass)
geom | geometry(LineString,4326) |

-- This will change the geometry columns to use constraints if they are not typmod or have constraints already.
--For this to work, there must exist data
CREATE TABLE public.myspatial_table_cs(gid serial, geom geometry);
INSERT INTO myspatial_table_cs(geom) VALUES(ST_GeomFromText('LINESTRING(1 2, 3 4)',4326));
SELECT Populate_Geometry_Columns('public.myspatial_table_cs'::regclass, false);
populate_geometry_columns

1
\d myspatial_table_cs

Table "public.myspatial_table_cs"
Column | Type | Modifiers
-----+-----+-----
gid | integer | not null default nextval('myspatial_table_cs_gid_seq'::regclass)
geom | geometry |
Check constraints:
 "enforce_dims_geom" CHECK (st_ndims(geom) = 2)
 "enforce_geotype_geom" CHECK (geometrytype(geom) = 'LINESTRING'::text OR geom IS NULL)
 "enforce_srid_geom" CHECK (st_srid(geom) = 4326)
```

7.2.6 UpdateGeometrySRID

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

Synopsis

```
text UpdateGeometrySRID(varchar table_name, varchar column_name, integer srid);
text UpdateGeometrySRID(varchar schema_name, varchar table_name, varchar column_name, integer srid);
text UpdateGeometrySRID(varchar catalog_name, varchar schema_name, varchar table_name, varchar column_name, integer srid);
```

Descripción

Actualiza el SRID de todos los registros de una columna de geometrías, actualizando las restricciones y referencias en geometry\_columns. Nota: utiliza current\_schema() en instalaciones postgresql que aceptan esquemas, si no se pasa ningún esquema.

-  This function supports 3d and will not drop the z-index.
-  This method supports Circular Strings and Curves.

Ejemplos

Insert geometries into roads table with a SRID set already using **EWKT format**:

```
COPY roads (geom) FROM STDIN;
SRID=4326;LINESTRING(0 0, 10 10)
SRID=4326;LINESTRING(10 10, 15 0)
\.
```

Esto cambiará el srid de la tabla de roads a 4326 de lo que era antes

```
SELECT UpdateGeometrySRID('roads','geom',4326);
```

El ejemplo previo es equivalente a esta sentencia DDL

```
ALTER TABLE roads
 ALTER COLUMN geom TYPE geometry(MULTILINESTRING, 4326)
 USING ST_SetSRID(geom,4326);
```

Si se obtuvo la proyección incorrecta (o que se señala como desconocido) en la carga y que quería transformar a mercator web todo en una sola toma, puede hacer esto con DDL pero no hay ninguna función de gestión de PostGIS equivalente para hacerlo de una sola vez.

```
ALTER TABLE roads
 ALTER COLUMN geom TYPE geometry(MULTILINESTRING, 3857) USING ST_Transform(ST_SetSRID(geom ←
 ,4326),3857) ;
```

También puedes ver

[UpdateRasterSRID](#), [ST\\_SetSRID](#), [ST\\_Transform](#)

## 7.3 Constructores Geométricos

### 7.3.1 ST\_GeomCollFromText

ST\_GeomCollFromText — Creates a GeometryCollection or Multi\* geometry from a set of geometries.

#### Synopsis

```
geometry ST_Collect(geometry g1, geometry g2);
geometry ST_Collect(geometry[] g1_array);
geometry ST_Collect(geometry set g1field);
```

#### Descripción

Collects geometries into a geometry collection. The result is either a Multi\* or a GeometryCollection, depending on whether the input geometries have the same or different types (homogeneous or heterogeneous). The input geometries are left unchanged within the collection.

**Variant 1:** accepts two input geometries

**Variant 2:** accepts an array of geometries

**Variant 3:** aggregate function accepting a rowset of geometries.



#### Note

If any of the input geometries are collections (Multi\* or GeometryCollection) ST\_Collect returns a GeometryCollection (since that is the only type which can contain nested collections). To prevent this, use [ST\\_Dump](#) in a subquery to expand the input collections to their atomic elements (see example below).

**Note**

ST\_Collect and ST\_Union appear similar, but in fact operate quite differently. ST\_Collect aggregates geometries into a collection without changing them in any way. ST\_Union geometrically merges geometries where they overlap, and splits linestrings at intersections. It may return single geometries when it dissolves boundaries.

Disponibilidad: 1.4.0 - ST\_MakeLine (geomarray) fue introducido. Las Funciones agregadas ST\_MakeLine se mejoraron para manejar más puntos más rápido.



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



This method supports Circular Strings and Curves.

**Ejemplos - uso de XLink**

Collect 2D points.

```
SELECT ST_AsText(ST_Collect(ST_GeomFromText('POINT(1 2)'),
 ST_GeomFromText('POINT(-2 3)')));
```

```
st_astext

MULTIPOINT((1 2), (-2 3))
```

Collect 3D points.

```
SELECT ST_AsEWKT(ST_Collect(ST_GeomFromEWKT('POINT(1 2 3)'),
 ST_GeomFromEWKT('POINT(1 2 4)')));
```

```
st_asewkt

MULTIPOINT(1 2 3, 1 2 4)
```

Collect curves.

```
SELECT ST_AsText(ST_Collect('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)',
 'CIRCULARSTRING(220227 150406,220227 150407,220227 150406)'));
```

```
st_astext

MULTICURVE(CIRCULARSTRING(220268 150415,220227 150505,220227 150406),
CIRCULARSTRING(220227 150406,220227 150407,220227 150406))
```

**Ejemplos: Utilizando la versión Array**

Using an array constructor for a subquery.

```
SELECT ST_Collect(ARRAY(SELECT geom FROM sometable));
```

Using an array constructor for values.

```
SELECT ST_AsText(ST_Collect(
 ARRAY[ST_GeomFromText('LINESTRING(1 2, 3 4)'),
 ST_GeomFromText('LINESTRING(3 4, 4 5)')])) As wktcollect;
```

```
--wkt collect --
MULTILINESTRING((1 2, 3 4), (3 4, 4 5))
```

## Ejemplos: Version Agregado Espacial

Creating multiple collections by grouping geometries in a table.

```
SELECT stusps, ST_Collect(f.geom) as geom
 FROM (SELECT stusps, (ST_Dump(geom)).geom As geom
 FROM
 somestatetable) As f
 GROUP BY stusps
```

Ver también

[ST\\_Dump](#), [ST\\_AsBinary](#)

### 7.3.2 ST\_LineFromMultiPoint

**ST\_LineFromMultiPoint** — Crea una LineString desde una geometría MultiPoint.

#### Synopsis

geometry **ST\_LineFromMultiPoint**(geometry aMultiPoint);

#### Descripción

Crea una LineString desde una geometría MultiPoint.

Use [ST\\_MakeLine](#) to create lines from Point or LineString inputs.



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

#### Ejemplos

Crea una LineString desde una geometría MultiPoint.

```
SELECT ST_AsEWKT(ST_LineFromMultiPoint('MULTIPOINT(1 2 3, 4 5 6, 7 8 9)'));

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

Ver también

[ST\\_AsEWKT](#), [ST\\_AsKML](#)

### 7.3.3 ST\_MakeEnvelope

**ST\_MakeEnvelope** — Crea un polígono rectangular formado a partir de los mínimos y máximos especificados. Los valores de entrada deben estar en el SRS especificado en el SRID.

#### Synopsis

geometry **ST\_MakeEnvelope**(double precision xmin, double precision ymin, double precision xmax, double precision ymax, integer srid=unknown);

**Descripción**

Crea un polígono rectangular formado a partir de los mínimos y máximos de la caja dada. Los valores de entrada deben estar en el SRS especificado por el SRID. Si no se especifica SRID se supone que el sistema de referencia espacial es desconocido.

Disponibilidad: 1.5

Mejorado: 2.0: Se introdujo capacidad de especificar una caja sin especificar un SRID.

**Ejemplo: Contruir un poligono correspondiente a la bounding box**

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

st_asewkt

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

**Ver también**

[ST\\_MakePoint](#), [ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_SRID](#)

**7.3.4 ST\_MakeLine**

ST\_MakeLine — Crea una cadena de línea desde geometrías de punto, multipunto o de línea.

**Synopsis**

```
geometry ST_MakeLine(geometry geom1, geometry geom2);
geometry ST_MakeLine(geometry[] geoms_array);
geometry ST_MakeLine(geometry set geoms);
```

**Descripción**

Creates a LineString containing the points of Point, MultiPoint, or LineString geometries. Other geometry types cause an error.

**Variant 1:** accepts two input geometries

**Variant 2:** accepts an array of geometries

**Variant 3:** aggregate function accepting a rowset of geometries. To ensure the order of the input geometries use `ORDER BY` in the function call, or a subquery with an `ORDER BY` clause.

Repeated nodes at the beginning of input LineStrings are collapsed to a single point. Repeated points in Point and MultiPoint inputs are not collapsed. [ST\\_RemoveRepeatedPoints](#) can be used to collapse repeated points from the output LineString.



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

Disponibilidad: 2.3.0 - Se introdujo soporte para elementos de entrada multipunto

Disponibilidad: 2.0.0 - Se introdujo el soporte de una cadena lineal como elemento de entrada

Disponibilidad: 1.4.0 - ST\_MakeLine (geomarray) fue introducido. Las Funciones agregadas ST\_MakeLine se mejoraron para manejar más puntos más rápido.

**Ejemplos: Utilizando la versión Array**

Create a line composed of two points.

```
SELECT ST_AsText(ST_MakeLine(ST_Point(1,2), ST_Point(3,4)));

 st_astext

LINESTRING(1 2,3 4)
```

Crea una BOX3D definida por las geometrías puntuales 2 3D dadas.

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

 st_asewkt

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

Crea una cadena de línea desde geometrías de punto, multipunto o de línea.

```
select ST_AsText(ST_MakeLine('LINESTRING(0 0, 1 1)', 'LINESTRING(2 2, 3 3)'));

 st_astext

LINESTRING(0 0,1 1,2 2,3 3)
```

**Ejemplos: Utilizando la versión Array**

Create a line from an array formed by a subquery with ordering.

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

Create a 3D line from an array of 3D points

```
SELECT ST_AsEWKT(ST_MakeLine(
 ARRAY[ST_MakePoint(1,2,3), ST_MakePoint(3,4,5), ST_MakePoint(6,6,6)]));

 st_asewkt

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

**Ejemplos: Version Agregado Espacial**

Este ejemplo toma una secuencia de puntos GPS y crea un registro para cada trayecto GPS donde el campo geometría es una cadena lineal compuesta de los puntos GPS en el orden del trayecto.

Using aggregate ORDER BY provides a correctly-ordered LineString.

```
SELECT gps.track_id, ST_MakeLine(gps.geom ORDER BY gps_time) As geom
FROM gps_points As gps
GROUP BY track_id;
```

Prior to PostgreSQL 9, ordering in a subquery can be used. However, sometimes the query plan may not respect the order of the subquery.

```
SELECT gps.track_id, ST_MakeLine(gps.geom) As geom
FROM (SELECT track_id, gps_time, geom
 FROM gps_points ORDER BY track_id, gps_time) As gps
GROUP BY track_id;
```



**Ver también**

[ST\\_RemoveRepeatedPoints](#), [ST\\_AsText](#), [ST\\_GeomFromText](#), [ST\\_MakePoint](#)

**7.3.5 ST\_MakePoint**

**ST\_MakePoint** — Creates a 2D, 3DZ or 4D Point.

**Synopsis**

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

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

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

**Descripción**

Crea una BOX2D definida por los puntos de la geometría dada.

Use [ST\\_MakePointM](#) to make points with XYM coordinates.

While not OGC-compliant, `ST_MakePoint` is faster and more precise than [ST\\_GeomFromText](#) and [ST\\_PointFromText](#). It is also easier to use for numeric coordinate values.

**Note**

For geodetic coordinates, X is longitude and Y is latitude



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

**Ejemplos**

```
--Return point with unknown SRID
SELECT ST_MakePoint(-71.1043443253471, 42.3150676015829);

--Return point marked as WGS 84 long lat
SELECT ST_SetSRID(ST_MakePoint(-71.1043443253471, 42.3150676015829), 4326);

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

--Get z of point
SELECT ST_Z(ST_MakePoint(1, 2, 1.5));
result

1.5
```

**Ver también**

[ST\\_GeomFromText](#), [ST\\_PointFromText](#), [ST\\_SetSRID](#), [ST\\_MakePointM](#)

### 7.3.6 ST\_MakePointM

ST\_MakePointM — Crea un punto con coordenadas x, y y un valor de medida.

#### Synopsis

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

#### Descripción

Crea un punto con coordenadas x, y y un valor de medida.

Use **ST\_MakePoint** to make points with XY, XYZ, or XYZM coordinates.



#### Note

For geodetic coordinates, X is longitude and Y is latitude

#### Ejemplos



#### Note

**ST\_AsEWKT** is used for text output because **ST\_AsText** does not support M values.

Create point with unknown SRID.

```
SELECT ST_AsEWKT(ST_MakePointM(-71.1043443253471, 42.3150676015829, 10));

 st_asewkt

POINTM(-71.1043443253471 42.3150676015829 10)
```

Crea un punto con coordenadas x, y y un valor de medida.

```
SELECT ST_AsEWKT(ST_SetSRID(ST_MakePointM(-71.104, 42.315, 10), 4326));

 st_asewkt

SRID=4326;POINTM(-71.104 42.315 10)
```

Get measure of created point.

```
SELECT ST_M(ST_MakePointM(-71.104, 42.315, 10));

result

10
```

#### Ver también

**ST\_AsEWKT**, **ST\_MakePoint**, **ST\_SetSRID**

### 7.3.7 ST\_MakePolygon

ST\_MakePolygon — Creates a Polygon from a shell and optional list of holes.

#### Synopsis

geometry **ST\_MakePolygon**(geometry linestring);

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

#### Descripción

Crea un polígono formado por el contorno dado. Las geometrías de entrada deben ser LINESTRINGS cerradas.

**Variant 1:** Accepts one shell LineString.

**Variant 2:** Accepts a shell LineString and an array of inner (hole) LineStrings. A geometry array can be constructed using the PostgreSQL array\_agg(), ARRAY[] or ARRAY() constructs.



#### Note

Esta función no acepta una MULTILINESTRING. Utiliza [ST\\_LineMerge](#) o [ST\\_Dump](#) para generar una linestring.



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

#### Ejemplos: Utilizando la versión Array

Crea un LineString desde una cadena de polilínea codificada.

```
SELECT ST_MakePolygon(ST_GeomFromText('LINESTRING(75 29,77 29,77 29, 75 29)'));
```

Create a Polygon from an open LineString, using [ST\\_StartPoint](#) and [ST\\_AddPoint](#) to close it.

```
SELECT ST_MakePolygon(ST_AddPoint(foo.open_line, ST_StartPoint(foo.open_line)))
FROM (
 SELECT ST_GeomFromText('LINESTRING(75 29,77 29,77 29, 75 29)') As open_line) As foo;
```

Crea un LineString desde una cadena de polilínea codificada.

```
SELECT ST_AsEWKT(ST_MakePolygon('LINESTRING(75.15 29.53 1,77 29 1,77.6 29.5 1, 75.15 ↵
 29.53 1)')));

st_asewkt

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

Create a Polygon from a LineString with measures

```
SELECT ST_AsEWKT(ST_MakePolygon('LINESTRINGM(75.15 29.53 1,77 29 1,77.6 29.5 2, 75.15 ↵
 29.53 2)')));

st_asewkt

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

## Ejemplos: carcasa exterior con carcadas interiores

Construye un donut con un agujero de hormiga

```
SELECT ST_MakePolygon(ST_ExteriorRing(ST_Buffer(ring.line,10)),
 ARRAY[ST_Translate(ring.line, 1, 1),
 ST_ExteriorRing(ST_Buffer(ST_Point(20,20),1))]
)
FROM (SELECT ST_ExteriorRing(
 ST_Buffer(ST_Point(10,10),10,10)) AS line) AS ring;
```

Create a set of province boundaries with holes representing lakes. The input is a table of province Polygons/MultiPolygons and a table of water linestrings. Lines forming lakes are determined by using **ST\_IsClosed**. The province linework is extracted by using **ST\_Boundary**. As required by **ST\_MakePolygon**, the boundary is forced to be a single LineString by using **ST\_LineMerge**. (However, note that if a province has more than one region or has islands this will produce an invalid polygon.) Using a **LEFT JOIN** ensures all provinces are included even if they have no lakes.



### Note

El constructor CASE se utiliza porque la alimentación de una matriz nula en **ST\_MakePolygon** resulta en NULL.

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

Another technique is to utilize a correlated subquery and the **ARRAY()** constructor that converts a row set to an array.

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

Ver también

**ST\_BuildArea** **ST\_Polygon**

## 7.3.8 ST\_Point

**ST\_Point** — Creates a Point with X, Y and SRID values.

## Synopsis

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

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

## Descripción

Returns a Point with the given X and Y coordinate values. This is the SQL-MM equivalent for **ST\_MakePoint** that takes just X and Y.



### Note

For geodetic coordinates, X is longitude and Y is latitude

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with **ST\_SetSRID** to mark the srid on the geometry.



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

## Ejemplos: Geometry

```
SELECT ST_Point(-71.104, 42.315);
```

```
SELECT ST_SetSRID(ST_Point(-71.104, 42.315),4326);
```

New in 3.2.0: With SRID specified

```
SELECT ST_Point(-71.104, 42.315, 4326);
```

## Ejemplos: Geography

Pre-PostGIS 3.2 syntax

```
SELECT CAST(ST_SetSRID(ST_Point(-71.104, 42.315), 4326) AS geography);
```

3.2 and on you can include the srid

```
SELECT CAST(ST_Point(-71.104, 42.315, 4326) AS geography);
```

PostgreSQL also provides the `::` short-hand for casting

```
SELECT ST_Point(-71.104, 42.315, 4326)::geography;
```

If the point coordinates are not in a geodetic coordinate system (such as WGS84), then they must be reprojected before casting to a geography. In this example a point in Pennsylvania State Plane feet (SRID 2273) is projected to WGS84 (SRID 4326).

```
SELECT ST_Transform(ST_SetSRID(ST_Point(3637510, 3014852), 2273), 4326)::geography;
```

## Ver también

Section 4.3, **ST\_MakePoint**, **ST\_SetSRID**, **ST\_Transform**, **ST\_Point**, **ST\_Point**, **ST\_Point**

### 7.3.9 ST\_Point

**ST\_Point** — Creates a Point with X, Y, Z and SRID values.

#### Synopsis

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

#### Descripción

Devuelve un **ST\_Point** con el valor de coordenadas dado. Es un alias de **ST\_MakePoint** del OGC.

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with **ST\_SetSRID** to mark the srid on the geometry.

#### Ejemplos

```
SELECT ST_PointZ(-71.104, 42.315, 3.4, 4326)
```

```
SELECT ST_PointZ(-71.104, 42.315, 3.4, srid =
> 4326)
```

```
SELECT ST_PointZ(-71.104, 42.315, 3.4)
```

#### Ver también

[ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_SetSRID](#), [ST\\_SRID](#)

### 7.3.10 ST\_Point

**ST\_Point** — Creates a Point with X, Y, M and SRID values.

#### Synopsis

geometry **ST\_PointM**(float x, float y, float m, integer srid=unknown);

#### Descripción

Devuelve un **ST\_Point** con el valor de coordenadas dado. Es un alias de **ST\_MakePoint** del OGC.

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with **ST\_SetSRID** to mark the srid on the geometry.

#### Ejemplos

```
SELECT ST_PointM(-71.104, 42.315, 3.4, 4326)
```

```
SELECT ST_PointM(-71.104, 42.315, 3.4, srid =
> 4326)
```

```
SELECT ST_PointM(-71.104, 42.315, 3.4)
```

**Ver también**

[ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_SetSRID](#), [ST\\_SRID](#)

### 7.3.11 ST\_Point

**ST\_Point** — Creates a Point with X, Y, Z, M and SRID values.

**Synopsis**

geometry **ST\_MakeEnvelope**(double precision xmin, double precision ymin, double precision xmax, double precision ymax, integer srid=unknown);

**Descripción**

Devuelve un **ST\_Point** con el valor de coordenadas dado. Es un alias de **ST\_MakePoint** del OGC.

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with **ST\_SetSRID** to mark the srid on the geometry.

**Ejemplos**

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5, 4326)
```

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5, srid =
> 4326)
```

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5)
```

**Ver también**

[ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_Point](#), [ST\\_Point](#), [ST\\_SetSRID](#)

### 7.3.12 ST\_Polygon

**ST\_Polygon** — Creates a Polygon from a LineString with a specified SRID.

**Synopsis**

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

**Descripción**

Returns a polygon built from the given LineString and sets the spatial reference system from the `srid`.

**ST\_Polygon** is similar to [ST\\_MakePolygon](#) Variant 1 with the addition of setting the SRID.

, [ST\\_MakePoint](#), [ST\\_SetSRID](#)

---

**Note**

Esta función no acepta una MULTILINESTRING. Utiliza [ST\\_LineMerge](#) o [ST\\_Dump](#) para generar una linestring.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 8.3.2



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

**Ejemplos**

Create a 2D polygon.

```
SELECT ST_AsText(ST_Polygon('LINESTRING(75 29, 77 29, 77 29, 75 29)::geometry, 4326));
-- result --
POLYGON((75 29, 77 29, 77 29, 75 29))
```

Create a 3D polygon.

```
SELECT ST_AsEWKT(ST_Polygon(ST_GeomFromEWKT('LINESTRING(75 29 1, 77 29 2, 77 29 3, 75 29 1)')), 4326));
-- result --
SRID=4326;POLYGON((75 29 1, 77 29 2, 77 29 3, 75 29 1))
```

**Ver también**

[ST\\_AsEWKT](#), [ST\\_AsText](#), [ST\\_GeomFromEWKT](#), [ST\\_GeomFromText](#), [ST\\_LineMerge](#), [ST\\_MakePolygon](#)

**7.3.13 ST\_MakeEnvelope**

**ST\_MakeEnvelope** — Creates a rectangular Polygon in [Web Mercator](#) (SRID:3857) using the [XYZ tile system](#).

**Synopsis**

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

**Descripción**

Creates a rectangular Polygon giving the extent of a tile in the [XYZ tile system](#). The tile is specified by the zoom level Z and the XY index of the tile in the grid at that level. Can be used to define the tile bounds required by [ST\\_AsMVTGeom](#) to convert geometry into the MVT tile coordinate space.

By default, the tile envelope is in the [Web Mercator](#) coordinate system (SRID:3857) using the standard range of the Web Mercator system (-20037508.342789, 20037508.342789). This is the most common coordinate system used for MVT tiles. The optional `bounds` parameter can be used to generate tiles in any coordinate system. It is a geometry that has the SRID and extent of the "Zoom Level zero" square within which the XYZ tile system is inscribed.

The optional `margin` parameter can be used to expand a tile by the given percentage. E.g. `margin=0.125` expands the tile by 12.5%, which is equivalent to `buffer=512` when the tile extent size is 4096, as used in [ST\\_AsMVTGeom](#). This is useful to create a tile buffer to include data lying outside of the tile's visible area, but whose existence affects the tile rendering. For example, a



city name (a point) could be near an edge of a tile, so its label should be rendered on two tiles, even though the point is located in the visible area of just one tile. Using expanded tiles in a query will include the city point in both tiles. Use a negative value to shrink the tile instead. Values less than -0.5 are prohibited because that would eliminate the tile completely. Do not specify a margin when using with `ST_AsMVTGeom`. See the example for `ST_AsMVT`.

Mejorada: 2.0.0 se agregó el parámetro por defecto opcional `srid`.

Disponibilidad: 2.1.0

#### Ejemplo: Contruir un poligono correspondiente a la bounding box

```
SELECT ST_AsText(ST_TileEnvelope(2, 1, 1));

 st_astext

POLYGON((-10018754.1713945 0,-10018754.1713945 10018754.1713945,0 10018754.1713945,0 ↵
0,-10018754.1713945 0))

SELECT ST_AsText(ST_TileEnvelope(3, 1, 1, ST_MakeEnvelope(-180, -90, 180, 90, 4326)));

 st_astext

POLYGON((-135 45,-135 67.5,-90 67.5,-90 45,-135 45))
```

#### Ver también

`ST_MakeEnvelope`

### 7.3.14 ST\_HexagonGrid

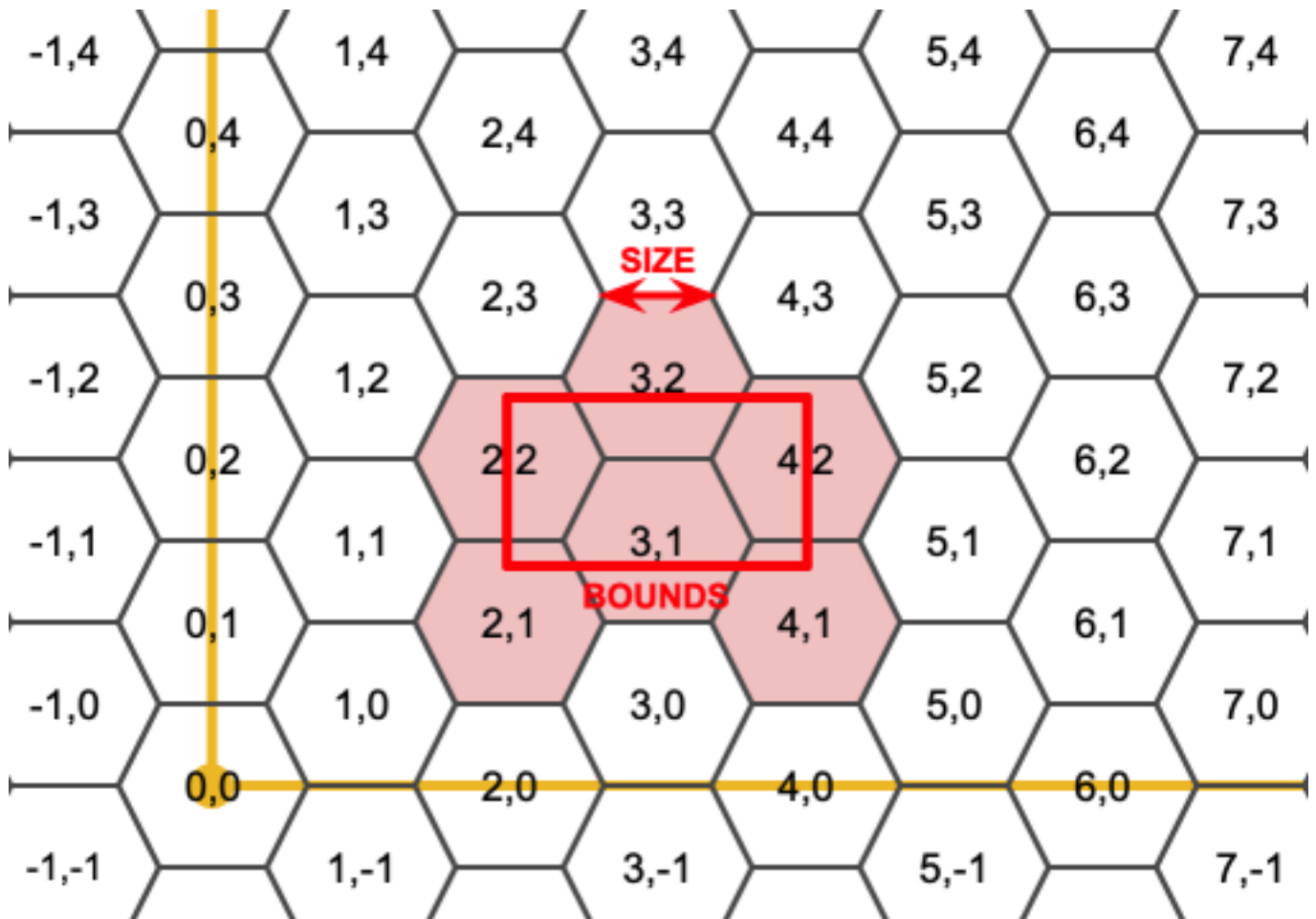
`ST_HexagonGrid` — Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.

#### Synopsis

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

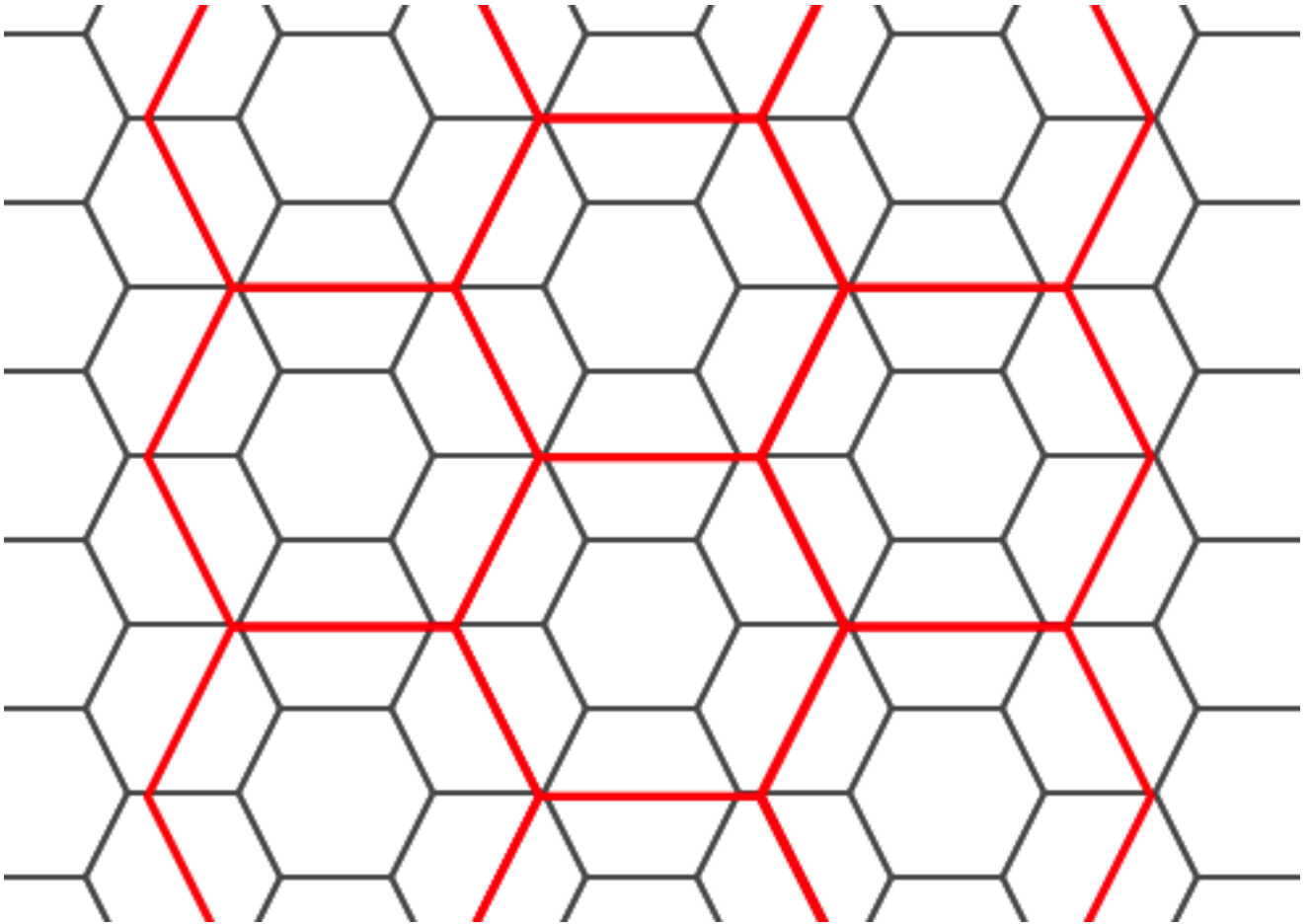
#### Descripción

Starts with the concept of a hexagon tiling of the plane. (Not a hexagon tiling of the globe, this is not the `H3` tiling scheme.) For a given planar SRS, and a given edge size, starting at the origin of the SRS, there is one unique hexagonal tiling of the plane, `Tiling(SRS, Size)`. This function answers the question: what hexagons in a given `Tiling(SRS, Size)` overlap with a given bounds.



The SRS for the output hexagons is the SRS provided by the bounds geometry.

Doubling or tripling the edge size of the hexagon generates a new parent tiling that fits with the origin tiling. Unfortunately, it is not possible to generate parent hexagon tilings that the child tiles perfectly fit inside.



Disponibilidad: 2.1.0

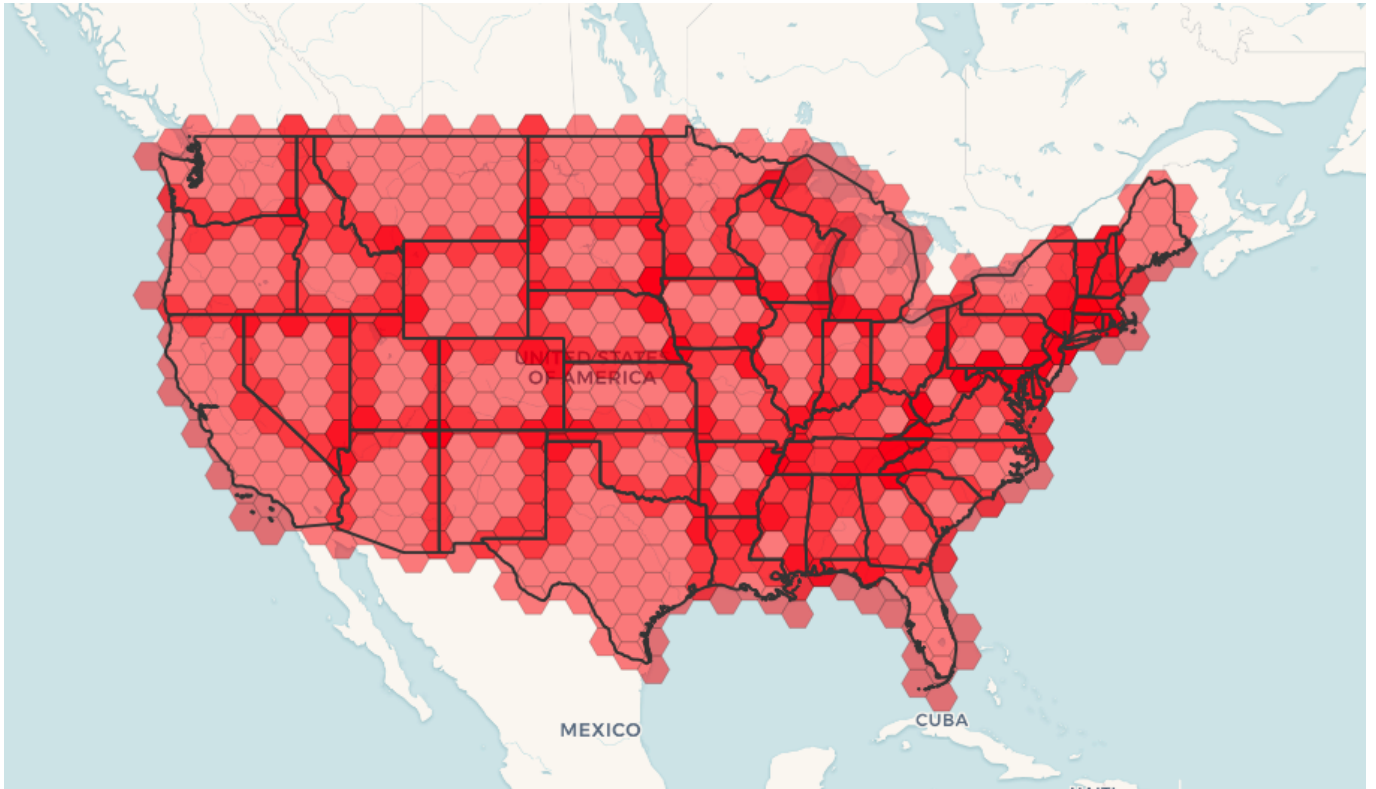
### Ejemplos: Utilizando la versión Array

To do a point summary against a hexagonal tiling, generate a hexagon grid using the extent of the points as the bounds, then spatially join to that grid.

```
SELECT COUNT(*), hexes.geom
FROM
 ST_HexagonGrid(
 10000,
 ST_SetSRID(ST_EstimatedExtent('pointtable', 'geom'), 3857)
) AS hexes
 INNER JOIN
 pointtable AS pts
 ON ST_Intersects(pts.geom, hexes.geom)
GROUP BY hexes.geom;
```

### Ejemplo: Contruir un poligono correspondiente a la bounding box

If we generate a set of hexagons for each polygon boundary and filter out those that do not intersect their hexagons, we end up with a tiling for each polygon.



Tiling states results in a hexagon coverage of each state, and multiple hexagons overlapping at the borders between states.



#### Note

The LATERAL keyword is implied for set-returning functions when referring to a prior table in the FROM list. So CROSS JOIN LATERAL, CROSS JOIN, or just plain , are equivalent constructs for this example.

```
SELECT admin1.gid, hex.geom
FROM
 admin1
 CROSS JOIN
 ST_HexagonGrid(100000, admin1.geom) AS hex
WHERE
 adm0_a3 = 'USA'
 AND
 ST_Intersects(admin1.geom, hex.geom)
```

#### Ver también

[ST\\_EstimatedExtent](#), [ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_SRID](#)

### 7.3.15 ST\_Hexagon

**ST\_Hexagon** — Returns a single hexagon, using the provided edge size and cell coordinate within the hexagon grid space.

#### Synopsis

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

**Descripción**

Uses the same hexagon tiling concept as [ST\\_HexagonGrid](#), but generates just one hexagon at the desired cell coordinate. Optionally, can adjust origin coordinate of the tiling, the default origin is at 0,0.

Hexagons are generated with no SRID set, so use [ST\\_SetSRID](#) to set the SRID to the one you expect.

Disponibilidad: 2.1.0

**Example: Creating a hexagon at the origin**

```
SELECT ST_AsText(ST_SetSRID(ST_Hexagon(1.0, 0, 0), 3857));

POLYGON((-1 0,-0.5
 -0.866025403784439,0.5
 -0.866025403784439,1
 0,0.5
 0.866025403784439,-0.5
 0.866025403784439,-1 0))
```

**Ver también**

[ST\\_MakeEnvelope](#), [ST\\_MakePoint](#), [ST\\_SetSRID](#)

**7.3.16 ST\_SquareGrid**

**ST\_SquareGrid** — Returns a set of grid squares and cell indices that completely cover the bounds of the geometry argument.

**Synopsis**

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

**Descripción**

Starts with the concept of a square tiling of the plane. For a given planar SRS, and a given edge size, starting at the origin of the SRS, there is one unique square tiling of the plane, `Tiling(SRS, Size)`. This function answers the question: what grids in a given `Tiling(SRS, Size)` overlap with a given bounds.

The SRS for the output squares is the SRS provided by the bounds geometry.

Doubling or edge size of the square generates a new parent tiling that perfectly fits with the original tiling. Standard web map tilings in mercator are just powers-of-two square grids in the mercator plane.

Disponibilidad: 2.1.0

**Example: Generating a 1 degree grid for a country**

The grid will fill the whole bounds of the country, so if you want just squares that touch the country you will have to filter afterwards with `ST_Intersects`.

```
WITH grid AS (
SELECT (ST_SquareGrid(1, ST_Transform(geom,4326))).*
FROM admin0 WHERE name = 'Canada'
)
SELEcT ST_AsText(geom)
FROM grid
```

**Example: Counting points in squares (using single chopped grid)**

To do a point summary against a square tiling, generate a square grid using the extent of the points as the bounds, then spatially join to that grid. Note the estimated extent might be off from actual extent, so be cautious and at very least make sure you've analyzed your table.

```
SELECT COUNT(*), squares.geom
FROM
 pointtable AS pts
 INNER JOIN
 ST_SquareGrid(
 1000,
 ST_SetSRID(ST_EstimatedExtent('pointtable', 'geom'), 3857)
) AS squares
ON ST_Intersects(pts.geom, squares.geom)
GROUP BY squares.geom
```

**Example: Counting points in squares using set of grid per point**

This yields the same result as the first example but will be slower for a large number of points

```
SELECT COUNT(*), squares.geom
FROM
 pointtable AS pts
 INNER JOIN
 ST_SquareGrid(
 1000,
 pts.geom
) AS squares
ON ST_Intersects(pts.geom, squares.geom)
GROUP BY squares.geom
```

**Ver también**

[ST\\_MakeEnvelope](#), [ST\\_Point](#), [ST\\_SetSRID](#), [ST\\_SRID](#)

### 7.3.17 ST\_Square

**ST\_Square** — Returns a single square, using the provided edge size and cell coordinate within the square grid space.

**Synopsis**

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

**Descripción**

Uses the same square tiling concept as [ST\\_SquareGrid](#), but generates just one square at the desired cell coordinate. Optionally, can adjust origin coordinate of the tiling, the default origin is at 0,0.

Squares are generated with no SRID set, so use [ST\\_SetSRID](#) to set the SRID to the one you expect.

Disponibilidad: 2.1.0

**Example: Creating a square at the origin**

```
SELECT ST_AsText(ST_SetSRID(ST_Square(1.0, 0, 0), 3857));

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

**Ver también**

[ST\\_MakeEnvelope](#), [ST\\_MakeLine](#), [ST\\_MakePolygon](#)

**7.3.18 ST\_Letters**

**ST\_Letters** — Returns the input letters rendered as geometry with a default start position at the origin and default text height of 100.

**Synopsis**

geometry **ST\_Letters**(text letters, json font);

**Descripción**

Uses a built-in font to render out a string as a multipolygon geometry. The default text height is 100.0, the distance from the bottom of a descender to the top of a capital. The default start position places the start of the baseline at the origin. Over-riding the font involves passing in a json map, with a character as the key, and base64 encoded TWKB for the font shape, with the fonts having a height of 1000 units from the bottom of the descenders to the tops of the capitals.

The text is generated at the origin by default, so to reposition and resize the text, first apply the `ST_Scale` function and then apply the `ST_Translate` function.

Disponibilidad: 2.1.0

**Ejemplo: Contruir un poligono correspondiente a la bounding box**

```
SELECT ST_AsText(ST_Letters('Yo'), 1);
```



*Letters generated by ST\_Letters*

**Example: Scaling and moving words**

```
SELECT ST_Translate(ST_Scale(ST_Letters('Yo'), 10, 10), 100,100);
```

**Ver también**

[ST\\_AsTWKB](#), [ST\\_Scale](#), [ST\\_Translate](#)

## 7.4 Métodos de Acceso a Geometrías

### 7.4.1 GeometryType

GeometryType — Devuelve el tipo de geometría del valor de ST\_Geometry.

**Synopsis**

text **GeometryType**(geometry geomA);

**Descripción**

Devuelve el tipo de geometría como una cadena de texto. Ej: 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.

OGC SPEC s2.1.1.1 - Devuelve el nombre del subtipo de la instancia de la geometría de la cual la instancia de la geometría es miembro. El nombre del subtipo de geometría de la instancia se devuelve en forma de cadena de texto.

**Note**

Esta función también indica si la geometría tiene valores de medida, devolviendo una cadena de tipo 'POINTM'.

Mejorado: 2.0.0 se introdujo soporte para superficies poliédricas, Triangulos y TIN.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method supports Circular Strings and Curves.



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



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**Ejemplos**

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

LINESTRING
```



```

SELECT ST_GeometryType(ST_GeomFromEWKT('POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0
0 0)),
 ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)
),
 ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
 ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)
))');
--result
POLYHEDRALSURFACE

```

```

SELECT GeometryType(geom) as result
FROM
 (SELECT
 ST_GeomFromEWKT('TIN (((
 0 0 0,
 0 0 1,
 0 1 0,
 0 0 0
)), ((
 0 0 0,
 0 1 0,
 1 1 0,
 0 0 0
))
)') AS geom
) AS g;
result

TIN

```

## Ver también

[ST\\_GeometryType](#)

## 7.4.2 ST\_Boundary

**ST\_Boundary** — Devuelve el cierre del limite combinatorio de esta geometría.

### Synopsis

geometry **ST\_Boundary**(geometry geomA);

### Descripción

Devuelve el cierre del limite combinatorio de esta geometría. El limite combinatorio esta definido como se describe en la sección 3.12.3.2 de la especificación OGC. Ya que el resultado de esta función es un cerco, y por lo tanto topológicamente cerrado, el límite resultante puede ser representado utilizando geometrías primitivas como se discute en la especificación OGC en la sección 3.12.2.

Realizado por el módulo de GEOS



#### Note

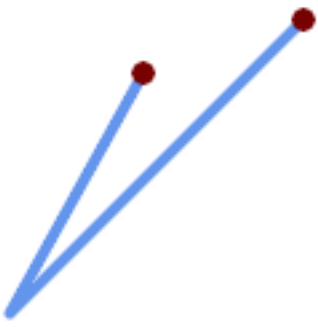
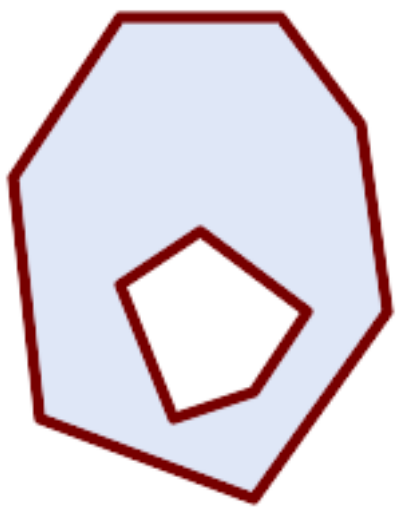
Anterior a la version 2.0.0, esta función lanza una excepción si se utiliza con `GEOMETRYCOLLECTION`. Desde la version 2.0.0 y superiores devolverá `NULL` en lugar de la excepción (entrada no soportada).

- ✓ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). OGC SPEC s2.1.1.1
- ✓ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.17
- ✓ This function supports 3d and will not drop the z-index.

Mejorado: 2.1.0 Se ha introducido soporte para Triangle

Changed: 3.2.0 support for TIN, does not use geos, does not linearize curves

## Ejemplos

|                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  <p><i>LineString con puntos de límite superpuestos</i></p> <pre>SELECT ST_Boundary(geom) FROM (SELECT 'LINESTRING(100 150,50 60, ↵ 70 80, 160 170) '::geometry As geom) As f;  ST_AsText output  MULTIPOINT((100 150),(160 170))</pre> |  <p><i>Agujeros de polígono con límite multilinestring</i></p> <pre>SELECT ST_Boundary(geom) FROM (SELECT 'POLYGON (( 10 130, 50 190, 110 190, 140 ↵ 150, 150 80, 100 10, 20 40, 10 130 ), ( 70 40, 100 50, 120 80, 80 110, ↵ 50 90, 70 40 ))'::geometry As geom) As f;  ST_AsText output  MULTILINESTRING((10 130,50 190,110 ↵ 190,140 150,150 80,100 10,20 40,10 130), (70 40,100 50,120 80,80 110,50 ↵ 90,70 40))</pre> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```
SELECT ST_AsText(ST_Boundary(ST_GeomFromText('LINESTRING(1 1,0 0, -1 1)')));
st_astext

MULTIPOINT((1 1),(-1 1))

SELECT ST_AsText(ST_Boundary(ST_GeomFromText('POLYGON((1 1,0 0, -1 1, 1 1))')));
st_astext

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

--Using a 3d polygon
```

```
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromEWKT('POLYGON((1 1 1,0 0 1, -1 1 1, 1 1 1))')));

st_asewkt

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

--Using a 3d multilinestring
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromEWKT('MULTILINESTRING((1 1 1,0 0 0.5, -1 1 1),(1 1 0.5,0 0 0.5, -1 1 0.5, 1 1 0.5))')));

st_asewkt

MULTIPOINT((-1 1 1),(1 1 0.75))
```

### Ver también

[ST\\_AsText](#), [ST\\_ExteriorRing](#), [ST\\_MakePolygon](#)

## 7.4.3 ST\_BoundingDiagonal

**ST\_BoundingDiagonal** — Devuelve la diagonal del cuadro delimitador de la geometría suministrada.

### Synopsis

geometry **ST\_BoundingDiagonal**(geometry geom, boolean fits=false);

### Descripción

Devuelve la diagonal del cuadro delimitador de la geometría suministrada como una cadena de línea. Si la geometría de entrada está vacía, la línea diagonal también está vacía, de lo contrario es una cadena de línea de 2 puntos con valores mínimos de cada dimensión en su punto de inicio y valores máximos en su punto final.

El parámetro *fits* especifica si se necesita el mejor ajuste. Si es false, se puede aceptar la diagonal de un cuadro delimitador algo más grande (es más rápido para obtener geometrías con muchos vértices). En cualquier caso, el cuadro delimitador de la línea diagonal devuelta siempre cubre la geometría de entrada.

La geometría cadena de línea devuelta siempre conserva SRID y dimensionalidad (z y m presente) de la geometría de entrada.



#### Note

En los casos degenerados (un solo vértice en la entrada) la cadena de líneas devuelta será topológicamente inválida (no interior). Esto no hace que el retorno sea semánticamente inválido.

Disponibilidad: 2.2.0



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



This function supports M coordinates.

## Ejemplos

```
-- Get the minimum X in a buffer around a point
SELECT ST_X(ST_StartPoint(ST_BoundingDiagonal(
 ST_Buffer(ST_Point(0,0),10)
)));
 st_x

 -10
```

## Ver también

[ST\\_StartPoint](#), [ST\\_EndPoint](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_Z](#), [ST\\_M](#), &&&

## 7.4.4 ST\_CoordDim

ST\_CoordDim — Devuelve la dimensión de las coordenadas del valor de ST\_Geometry.

## Synopsis

integer **ST\_CoordDim**(geometry geomA);

## Descripción

Devuelve la dimensión de las coordenadas del valor de ST\_Geometry.

Es el alias de [ST\\_NDims](#) conforme a MM



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 5.1.3



This method supports Circular Strings and Curves.



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



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Ejemplos

```
SELECT ST_CoordDim('CIRCULARSTRING(1 2 3, 1 3 4, 5 6 7, 8 9 10, 11 12 13)');
 ---result---
 3

 SELECT ST_CoordDim(ST_Point(1,2));
 --result--
 2
```

## Ver también

[ST\\_NDims](#)

### 7.4.5 ST\_Dimension

**ST\_Dimension** — Devuelve la dimensión de las coordenadas del valor de ST\_Geometry.

#### Synopsis

integer **ST\_Dimension**(geometry g);

#### Descripción

La dimensión inherente del objeto Geometry, la cual debe ser menor o igual a la dimensión de coordenadas. En la Especificación OGC s2.1.1.1 - devuelve 0 para un POINT, 1 para una LINESTRING, 2 para un POLYGON, y la dimensión mayor de los componentes de una GEOMETRYCOLLECTION. Si es desconocida (geometría vacía) se devuelve null.



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

Mejora: 2.0.0 se introdujeron soporte de superficies poliédricas y TIN. No lanza una excepción si se envía una geometría vacía.



#### Note

Anterior a la versión 2.0.0, esta función lanzaba una excepción si se enviaba una geometría vacía.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

#### Ejemplos

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

1
```

#### Ver también

[ST\\_NDims](#)

### 7.4.6 ST\_Dump

**ST\_Dump** — Returns a set of geometry\_dump rows for the components of a geometry.

#### Synopsis

geometry **ST\_Envelope**(geometry g1);

## Descripción

A set-returning function (SRF) that extracts the components of a geometry. It returns a set of **geometry\_dump** rows, each containing a geometry (*geom* field) and an array of integers (*path* field).

For an atomic geometry type (POINT,LINestring,POLYGON) a single record is returned with an empty *path* array and the input geometry as *geom*. For a collection or multi-geometry a record is returned for each of the collection components, and the *path* denotes the position of the component inside the collection.

ST\_Dump is useful for expanding geometries. It is the inverse of a **ST\_GeomCollFromText** / GROUP BY, in that it creates new rows. For example it can be use to expand MULTIPOLYGONS into POLYGONS.

Mejorado: 2.0.0 se introdujo soporte para superficies poliédricas, Triangulos y TIN.

Availability: PostGIS 1.0.0RC1. Requires PostgreSQL 7.3 or higher.



### Note

Anterior a 1.3.4, esta función daba errores si se utilizaba con geometrias que contenían CURVES. Esto se corrigió en 1.3.4+



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



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

## Ejemplos Estándar

```
SELECT sometable.field1, sometable.field1,
 (ST_Dump(sometable.geom)).geom AS geom
FROM sometable;

-- Break a compound curve into its constituent linestrings and circularstrings
SELECT ST_AsEWKT(a.geom), ST_HasArc(a.geom)
FROM (SELECT (ST_Dump(p_geom)).geom AS geom
 FROM (SELECT ST_GeomFromEWKT('COMPOUNDCURVE(CIRCULARSTRING(0 0, 1 1, 1 0),(1 0, 0 1))') AS p_geom) AS b
) AS a;
 st_asewkt | st_hasarc
-----+-----
CIRCULARSTRING(0 0,1 1,1 0) | t
LINestring(1 0,0 1) | f
(2 rows)
```

## Ejemplos de superficies poliedricas, MDT y triángulos

```
-- Polyhedral surface example
-- Break a Polyhedral surface into its faces
SELECT (a.p_geom).path[1] As path, ST_AsEWKT((a.p_geom).geom) As geom_ewkt
FROM (SELECT ST_Dump(ST_GeomFromEWKT('POLYHEDRALSURFACE(
((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)')) AS a);
```

```
)')) AS p_geom) AS a;

path | geom_ewkt
-----+-----
1 | POLYGON((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0))
2 | POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))
3 | POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0))
4 | POLYGON((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0))
5 | POLYGON((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0))
6 | POLYGON((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))

-- TIN --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
 (SELECT
 ST_Dump(ST_GeomFromEWKT('TIN (((
 0 0 0,
 0 0 1,
 0 1 0,
 0 0 0
)), ((
 0 0 0,
 0 1 0,
 1 1 0,
 0 0 0
))
)')) AS gdump
) AS g;
-- result --
path | wkt
-----+-----
{1} | TRIANGLE((0 0 0,0 0 1,0 1 0,0 0 0))
{2} | TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))
```

Ver también

[geometry\\_dump](#), [ST\\_GeomFromEWKT](#), [ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

7.4.7 ST\_NumPoints

ST\_NumPoints — Devuelve un resumen de texto del contenido de la geometría.

Synopsis

geometry **ST\_Points**( geometry geom );

Descripción

A set-returning function (SRF) that extracts the coordinates (vertices) of a geometry. It returns a set of [geometry\\_dump](#) rows, each containing a geometry (*geom* field) and an array of integers (*path* field).

- the *geom* field POINTs represent the coordinates of the supplied geometry.
- the *path* field (an integer[]) is an index enumerating the coordinate positions in the elements of the supplied geometry. The indices are 1-based. For example, for a LINESTRING the paths are {i} where i is the nth coordinate in the LINESTRING. For a POLYGON the paths are {i, j} where i is the ring number (1 is outer; inner rings follow) and j is the coordinate position in the ring.

To obtain a single geometry containing the coordinates use **ST\_Points**.

Enhanced: 2.1.0 Faster speed. Reimplemented as native-C.

Mejorado: 2.0.0 se introdujo soporte para superficies poliédricas, Triangulos y TIN.

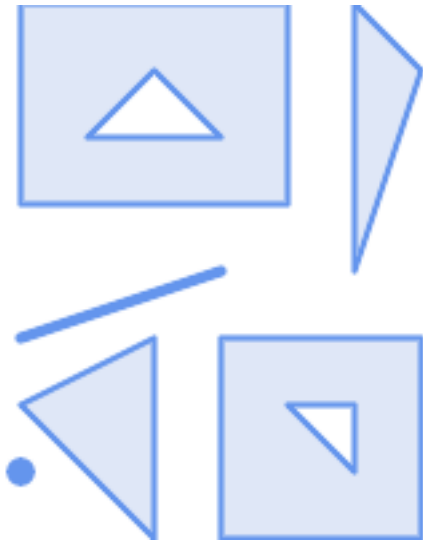
Disponibilidad: 1.2.2

- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✔ This function supports 3d and will not drop the z-index.

**Classic Explode a Table of LineStrings into nodes**

```
SELECT edge_id, (dp).path[1] As index, ST_AsText((dp).geom) As wktnode
FROM (SELECT 1 As edge_id
 , ST_DumpPoints(ST_GeomFromText('LINESTRING(1 2, 3 4, 10 10)')) AS dp
 UNION ALL
 SELECT 2 As edge_id
 , ST_DumpPoints(ST_GeomFromText('LINESTRING(3 5, 5 6, 9 10)')) AS dp
) As foo;
edge_id | index | wktnode
-----+-----+-----
1 | 1 | POINT(1 2)
1 | 2 | POINT(3 4)
1 | 3 | POINT(10 10)
2 | 1 | POINT(3 5)
2 | 2 | POINT(5 6)
2 | 3 | POINT(9 10)
```

**Ejemplos Estándar**



```
SELECT path, ST_AsText (geom)
FROM (
 SELECT (ST_DumpPoints(g.geom)) .*
 FROM
```



```

(SELECT
 'GEOMETRYCOLLECTION(
 POINT (0 1),
 LINESTRING (0 3, 3 4),
 POLYGON ((2 0, 2 3, 0 2, 2 0)),
 POLYGON ((3 0, 3 3, 6 3, 6 0, 3 0),
 (5 1, 4 2, 5 2, 5 1)),
 MULTIPOLYGON (
 ((0 5, 0 8, 4 8, 4 5, 0 5)),
 (1 6, 3 6, 2 7, 1 6)),
 ((5 4, 5 8, 6 7, 5 4))
)
)::geometry AS geom
) AS g
) j;

```

| path      | st_astext  |
|-----------|------------|
| {1,1}     | POINT(0 1) |
| {2,1}     | POINT(0 3) |
| {2,2}     | POINT(3 4) |
| {3,1,1}   | POINT(2 0) |
| {3,1,2}   | POINT(2 3) |
| {3,1,3}   | POINT(0 2) |
| {3,1,4}   | POINT(2 0) |
| {4,1,1}   | POINT(3 0) |
| {4,1,2}   | POINT(3 3) |
| {4,1,3}   | POINT(6 3) |
| {4,1,4}   | POINT(6 0) |
| {4,1,5}   | POINT(3 0) |
| {4,2,1}   | POINT(5 1) |
| {4,2,2}   | POINT(4 2) |
| {4,2,3}   | POINT(5 2) |
| {4,2,4}   | POINT(5 1) |
| {5,1,1,1} | POINT(0 5) |
| {5,1,1,2} | POINT(0 8) |
| {5,1,1,3} | POINT(4 8) |
| {5,1,1,4} | POINT(4 5) |
| {5,1,1,5} | POINT(0 5) |
| {5,1,2,1} | POINT(1 6) |
| {5,1,2,2} | POINT(3 6) |
| {5,1,2,3} | POINT(2 7) |
| {5,1,2,4} | POINT(1 6) |
| {5,2,1,1} | POINT(5 4) |
| {5,2,1,2} | POINT(5 8) |
| {5,2,1,3} | POINT(6 7) |
| {5,2,1,4} | POINT(5 4) |

(29 rows)

### Ejemplos de superficies poliedricas, MDT y triángulos

```

-- Polyhedral surface cube --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
 (SELECT
 ST_DumpPoints(ST_GeomFromEWKT('POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0) ←
 0)),
 ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
 ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
 ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))') AS gdump
)

```

```

) AS g;
-- result --
path | wkt
-----+-----
{1,1,1} | POINT(0 0 0)
{1,1,2} | POINT(0 0 1)
{1,1,3} | POINT(0 1 1)
{1,1,4} | POINT(0 1 0)
{1,1,5} | POINT(0 0 0)
{2,1,1} | POINT(0 0 0)
{2,1,2} | POINT(0 1 0)
{2,1,3} | POINT(1 1 0)
{2,1,4} | POINT(1 0 0)
{2,1,5} | POINT(0 0 0)
{3,1,1} | POINT(0 0 0)
{3,1,2} | POINT(1 0 0)
{3,1,3} | POINT(1 0 1)
{3,1,4} | POINT(0 0 1)
{3,1,5} | POINT(0 0 0)
{4,1,1} | POINT(1 1 0)
{4,1,2} | POINT(1 1 1)
{4,1,3} | POINT(1 0 1)
{4,1,4} | POINT(1 0 0)
{4,1,5} | POINT(1 1 0)
{5,1,1} | POINT(0 1 0)
{5,1,2} | POINT(0 1 1)
{5,1,3} | POINT(1 1 1)
{5,1,4} | POINT(1 1 0)
{5,1,5} | POINT(0 1 0)
{6,1,1} | POINT(0 0 1)
{6,1,2} | POINT(1 0 1)
{6,1,3} | POINT(1 1 1)
{6,1,4} | POINT(0 1 1)
{6,1,5} | POINT(0 0 1)
(30 rows)

```

```

-- Triangle --
SELECT (g.gdump).path, ST_AsText((g.gdump).geom) as wkt
FROM
 (SELECT
 ST_DumpPoints(ST_GeomFromEWKT('TRIANGLE ((
 0 0,
 0 9,
 9 0,
 0 0
))')) AS gdump
) AS g;
-- result --
path | wkt
-----+-----
{1} | POINT(0 0)
{2} | POINT(0 9)
{3} | POINT(9 0)
{4} | POINT(0 0)

```

```

-- TIN --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
 (SELECT
 ST_DumpPoints(ST_GeomFromEWKT('TIN (((
 0 0 0,

```

```

 0 0 1,
 0 1 0,
 0 0 0
), (
 0 0 0,
 0 1 0,
 1 1 0,
 0 0 0
)
) ') AS gdump
) AS g;
-- result --
path | wkt
-----+-----
{1,1,1} | POINT(0 0 0)
{1,1,2} | POINT(0 0 1)
{1,1,3} | POINT(0 1 0)
{1,1,4} | POINT(0 0 0)
{2,1,1} | POINT(0 0 0)
{2,1,2} | POINT(0 1 0)
{2,1,3} | POINT(1 1 0)
{2,1,4} | POINT(0 0 0)
(8 rows)
```

Ver también

[geometry\\_dump](#), [ST\\_GeomFromEWKT](#), [ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

7.4.8 ST\_NumPoints

ST\_NumPoints — Devuelve un resumen de texto del contenido de la geometría.

Synopsis

```
geometry ST_Points(geometry geom);
```

Descripción

A set-returning function (SRF) that extracts the segments of a geometry. It returns a set of [geometry\\_dump](#) rows, each containing a geometry (*geom* field) and an array of integers (*path* field).

- Devuelve TRUE si esta LINESTRING es simple y cerrada.
- the *path* field (an integer[]) is an index enumerating the segment start point positions in the elements of the supplied geometry. The indices are 1-based. For example, for a LINESTRING the paths are {*i*} where *i* is the *n*th segment start point in the LINESTRING. For a POLYGON the paths are {*i*, *j*} where *i* is the ring number (1 is outer; inner rings follow) and *j* is the segment start point position in the ring.

Disponibilidad: 2.2.0

-  This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
-  This function supports 3d and will not drop the z-index.

## Ejemplos Estándar

```
SELECT path, ST_AsText(geom)
FROM (
 SELECT (ST_DumpSegments(g.geom)).*
 FROM (SELECT 'GEOMETRYCOLLECTION(
 LINESTRING(1 1, 3 3, 4 4),
 POLYGON((5 5, 6 6, 7 7, 5 5))
)'::geometry AS geom
) AS g
) j;
```

| path    | &#x2502; | st_astext           |
|---------|----------|---------------------|
| {1,1}   | &#x2502; | LINESTRING(1 1,3 3) |
| {1,2}   | &#x2502; | LINESTRING(3 3,4 4) |
| {2,1,1} | &#x2502; | LINESTRING(5 5,6 6) |
| {2,1,2} | &#x2502; | LINESTRING(6 6,7 7) |
| {2,1,3} | &#x2502; | LINESTRING(7 7,5 5) |

(5 rows)

## Ejemplos de superficies poliedricas, MDT y triángulos

```
-- Triangle --
SELECT path, ST_AsText(geom)
FROM (
 SELECT (ST_DumpSegments(g.geom)).*
 FROM (SELECT 'TRIANGLE((
 0 0,
 0 9,
 9 0,
 0 0
)'::geometry AS geom
) AS g
) j;
```

| path  | &#x2502; | st_astext           |
|-------|----------|---------------------|
| {1,1} | &#x2502; | LINESTRING(0 0,0 9) |
| {1,2} | &#x2502; | LINESTRING(0 9,9 0) |
| {1,3} | &#x2502; | LINESTRING(9 0,0 0) |

(3 rows)

```
-- TIN --
SELECT path, ST_AsEWKT(geom)
FROM (
 SELECT (ST_DumpSegments(g.geom)).*
 FROM (SELECT 'TIN(((
 0 0 0,
 0 0 1,
 0 1 0,
 0 0 0
))), ((
 0 0 0,
 0 1 0,
 1 1 0,
 0 0 0
))
)'::geometry AS geom
) j;
```

```

) AS g
) j;

path │ st_asewkt

{1,1,1} │ LINESTRING(0 0 0,0 0 1)
{1,1,2} │ LINESTRING(0 0 1,0 1 0)
{1,1,3} │ LINESTRING(0 1 0,0 0 0)
{2,1,1} │ LINESTRING(0 0 0,0 1 0)
{2,1,2} │ LINESTRING(0 1 0,1 1 0)
{2,1,3} │ LINESTRING(1 1 0,0 0 0)
(6 rows)

```

### Ver también

[geometry\\_dump](#), [ST\\_GeomCollFromText](#), [ST\\_Dump](#), [ST\\_NumInteriorRing](#),

## 7.4.9 ST\_NRings

ST\_NRings — Returns a set of `geometry_dump` rows for the exterior and interior rings of a Polygon.

### Synopsis

geometry **ST\_ExteriorRing**(geometry a\_polygon);

### Descripción

A set-returning function (SRF) that extracts the rings of a polygon. It returns a set of [geometry\\_dump](#) rows, each containing a geometry (*geom* field) and an array of integers (*path* field).

The *geom* field contains each ring as a POLYGON. The *path* field is an integer array of length 1 containing the polygon ring index. The exterior ring (shell) has index 0. The interior rings (holes) have indices of 1 and higher.



#### Note

Esto no funcionara con MULTIPOLYGONS. Para MULTIPOLYGONS utilizaba junto a ST\_Dump.

Availability: PostGIS 1.1.3. Requires PostgreSQL 7.3 or higher.



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

### Ejemplos

General form of query.

```

SELECT polyTable.field1, polyTable.field1,
 (ST_DumpRings(polyTable.geom)).geom As geom
FROM polyTable;

```

A polygon with a single hole.

```

SELECT path, ST_AsEWKT(geom) As geom
FROM ST_DumpRings(
 ST_GeomFromEWKT('POLYGON((-8149064 5133092 1,-8149064 5132986 1,-8148996 5132839 1,-8148972 5132767 1,-8148958 5132508 1,-8148941 5132466 1,-8148924 5132394 1,-8148903 5132210 1,-8148930 5131967 1,-8148992 5131978 1,-8149237 5132093 1,-8149404 5132211 1,-8149647 5132310 1,-8149757 5132394 1,-8150305 5132788 1,-8149064 5133092 1),(-8149362 5132394 1,-8149446 5132501 1,-8149548 5132597 1,-8149695 5132675 1,-8149362 5132394 1))')
) as foo;

```

| path | geom                                                                                                                                                                                                                                                                                                                       |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| {0}  | POLYGON((-8149064 5133092 1,-8149064 5132986 1,-8148996 5132839 1,-8148972 5132767 1,-8148958 5132508 1,-8148941 5132466 1,-8148924 5132394 1,-8148903 5132210 1,-8148930 5131967 1,-8148992 5131978 1,-8149237 5132093 1,-8149404 5132211 1,-8149647 5132310 1,-8149757 5132394 1,-8150305 5132788 1,-8149064 5133092 1)) |
| {1}  | POLYGON((-8149362 5132394 1,-8149446 5132501 1,-8149548 5132597 1,-8149695 5132675 1,-8149362 5132394 1))                                                                                                                                                                                                                  |

## Ver también

[geometry\\_dump](#), [ST\\_GeomFromEWKT](#), [ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

## 7.4.10 ST\_EndPoint

**ST\_EndPoint** — Devuelve el número de puntos en un valor **ST\_LineString** o **ST\_CircularString**.

### Synopsis

geometry **ST\_Points**( geometry geom );

### Descripción

Devuelve el primer punto de una geometría **LINESTRING** o **CIRCULARLINESTRING** como un **POINT** o **NULL** si el parámetro de entrada no es un **LINESTRING** o **CIRCULARLINESTRING**.



This method implements the SQL/MM specification. SQL-MM 3: 7.1.4



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



This method supports Circular Strings and Curves.

### Note



Cambiado: 2.0.0 ya no funciona con multilinestrings de geometrías simples. En versiones anteriores de PostGIS -- una linea simple multilinestring funciona sin problemas con esta función y devuelve el punto inicial. En la version 2.0.0 simplemente devuelve NULL como con cualquier multilinestring. La antigua version era una función sin documentar, pero la gente que asumía que tenia sus datos almacenados en **LINESTRING** pueden experimentar este comportamiento ahora de resultado **NULL** en la version 2.0.

## Ejemplos

### End point of a LineString

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

POINT(3 3)
```

### End point of a non-LineString is NULL

```
SELECT ST_EndPoint('POINT(1 1)::geometry') IS NULL AS is_null;
is_null

t
```

### End point of a 3D LineString

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

POINT(0 0 5)
```

### Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString.

```
SELECT ST_AsText(ST_EndPoint('CIRCULARSTRING(5 2,-3 1.999999, -2 1, -4 2, 6 3)::geometry')) ←
;
st_astext

POINT(6 3)
```

## Ver también

[ST\\_PointN](#), [ST\\_StartPoint](#)

## 7.4.11 ST\_Envelope

**ST\_Envelope** — Devuelve una geometría que representa la caja en doble precisión (float8) de la geometría dada.

### Synopsis

geometry **ST\_Envelope**(geometry g1);

### Descripción

Devuelve una geometría que representa la caja mínima en doble precisión (float8) de la geometría dada. El polígono definido por las esquinas de la caja ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)). (PostGIS añadirá las coordenadas ZMIN/ZMAX también).

Algunos casos particulares (líneas verticales, puntos) devolverán una geometría de dimension menor que POLYGON, por ejemplo POINT o LINESTRING.

Disponibilidad: 1.5.0 comportamiento modificado para devolver doble precisión en vez de float4.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



This method implements the SQL/MM specification. SQL-MM 3: 5.1.19

## Ejemplos

```
SELECT ST_AsText(ST_Envelope('POINT(1 3)::geometry'));
 st_astext

POINT(1 3)
(1 row)

SELECT ST_AsText(ST_Envelope('LINESTRING(0 0, 1 3)::geometry'));
 st_astext

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

SELECT ST_AsText(ST_Envelope('POLYGON((0 0, 0 1, 1.0000001 1, 1.0000001 0, 0 0))::geometry' ↵
));
 st_astext

POLYGON((0 0,0 1,1.00000011920929 1,1.00000011920929 0,0 0))
(1 row)
SELECT ST_AsText(ST_Envelope('POLYGON((0 0, 0 1, 1.0000000001 1, 1.0000000001 0, 0 0)):: ↵
geometry'));
 st_astext

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

SELECT Box3D(geom), Box2D(geom), ST_AsText(ST_Envelope(geom)) As envelopewkt
FROM (SELECT 'POLYGON((0 0, 0 1000012333334.34545678, 1.0000001 1, 1.0000001 0, 0 ↵
0))::geometry As geom) As foo;
```



*Envelope of a point and linestring.*

```
SELECT ST_AsText(ST_Envelope(
 ST_Collect(
 ST_GeomFromText('LINESTRING(55 75,125 150)'),
 ST_Point(20, 80))
```



```

)) As wktenv;
wktenv

POLYGON((20 75,20 150,125 150,125 75,20 75))

```

### Ver también

[Box2D](#), [Box3D](#), [ST\\_OrientedEnvelope](#)

## 7.4.12 ST\_ExteriorRing

**ST\_ExteriorRing** — Devuelva el número de anillos interiores de una geometría poligonal.

### Synopsis

geometry **ST\_ExteriorRing**(geometry a\_polygon);

### Descripción

Devuelve una linestring representando el anillo exterior de una geometría tipo POLYGON. Devuelve NULL si la geometría no es un polígono. No funcionará con MULTIPOLYGON



#### Note

Esto no funcionara con MULTIPOLYGONS. Para MULTIPOLYGONS utilizaba junto a ST\_Dump.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). 2.1.5.1



This method implements the SQL/MM specification. SQL-MM 3: 8.2.3, 8.3.3



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

### Ejemplos

```

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

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

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

```

```
st_asewkt

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

### Ver también

[ST\\_InteriorRingN](#), [ST\\_Boundary](#), [ST\\_NumInteriorRings](#)

## 7.4.13 ST\_GeometryN

ST\_GeometryN — Devuelve el tipo de geometría del valor de ST\_Geometry.

### Synopsis

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

### Descripción

Devuelve la geometría en la cual se basa si la geometría es una GEOMETRYCOLLECTION, un (MULTI)POINT, una (MULTI)LINESTRING, una MULTICURVE o un (MULTI)POLYGON, una POLYHEDRALSURFACE si no devuelve NULL.



#### Note

El índice es 1-based en la especificación OGC desde la version 0.8.0. Versiones anteriormente implementadas era de tipo 0-based.



#### Note

Si quieres extraer todas las geometrías de una geometría, ST\_Dump es mas eficiente y funcionará con geometrías simples.

Mejorado: 2.0.0 se introdujo soporte para superficies poliédricas, Triangulos y TIN.

Cambiado: 2.0.0 Versiones anteriores devuelven NULL para geometrias simples. Esto ha sido cambiado para devolver la geometría en el caso de ST\_GeometryN(..,1) .



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 9.1.5



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



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Ejemplos Estándar

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

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

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

## Ejemplos de superficies poliedricas, MDT y triángulos

```
-- Polyhedral surface example
-- Break a Polyhedral surface into its faces
SELECT ST_AsEWKT(ST_GeometryN(p_geom,3)) As geom_ewkt
FROM (SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE(
((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)') AS p_geom) AS a;
```

| geom_ewkt                                |
|------------------------------------------|
| POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)) |

```
-- TIN --
SELECT ST_AsEWKT(ST_GeometryN(geom,2)) as wkt
FROM
(SELECT
ST_GeomFromEWKT('TIN (((
0 0 0,
0 0 1,
0 1 0,
0 0 0
)), ((
0 0 0,
0 1 0,
1 1 0,
0 0 0
)))
```

```

) ') AS geom
) AS g;
-- result --
 wkt

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

```

### Ver también

[ST\\_Dump](#), [ST\\_NumGeometries](#)

## 7.4.14 ST\_GeometryType

ST\_GeometryType — Devuelve el tipo de geometría del valor de ST\_Geometry.

### Synopsis

text **ST\_GeometryType**(geometry g1);

### Descripción

Devuelve el tipo de geometría como una cadena de texto. Por Ejemplo: 'ST\_LineString', 'ST\_Polygon', 'ST\_MultiPolygon' etc. Esta función difiere de GeometryType(geometría) en este caso se devuelve la cadena de texto y ST delante, como el hecho de que no indicará como se mide la geometría.

Mejora: 2.0.0 se introdujo soporte de superficies poliédricas.



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



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



This function supports Polyhedral surfaces.

### Ejemplos

```

SELECT ST_GeometryType(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
--result
ST_LineString

```

```

SELECT ST_GeometryType(ST_GeomFromEWKT('POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
))'));
--result
ST_PolyhedralSurface

```

```

SELECT ST_GeometryType(ST_GeomFromEWKT('POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
 ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
 ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
 ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
))');
--result
ST_PolyhedralSurface

```

```

SELECT ST_GeometryType(geom) as result
FROM
 (SELECT
 ST_GeomFromEWKT('TIN (((
 0 0 0,
 0 0 1,
 0 1 0,
 0 0 0
)), ((
 0 0 0,
 0 1 0,
 1 1 0,
 0 0 0
))
)') AS geom
) AS g;
result

ST_Tin

```

## Ver también

[GeometryType](#)

### 7.4.15 ST\_HasArc

ST\_HasArc — Tests if a geometry contains a circular arc

#### Synopsis

boolean **ST\_IsEmpty**(geometry geomA);

#### Descripción

Devuelve True si la Geometría es una colección vacía, polígono vacío, punto vacío etc.

Disponibilidad: 1.2.2



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



This method supports Circular Strings and Curves.

## Ejemplos

```
SELECT ST_HasArc(ST_Collect('LINESTRING(1 2, 3 4, 5 6)', 'CIRCULARSTRING(1 1, 2 3, 4 5, 6 6, 7, 5 6)'));
 st_hasarc
 -
 t
```

## Ver también

[ST\\_CurveToLine](#), [ST\\_PointN](#)

## 7.4.16 ST\_InteriorRingN

**ST\_InteriorRingN** — Devuelva el número de anillos interiores de una geometría poligonal.

### Synopsis

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

### Descripción

Devuelve la cadena de texto del anillo interior N del poligono. Devuelve NULL si la geometría no es un polígono o el índice N dado esta fuera de rango.



#### Note

Esto no funcionara con MULTIPOLYGONS. Para MULTIPOLYGONS utilizaba junto a ST\_Dump.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5



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

## Ejemplos

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

## Ver también

[ST\\_ExteriorRing](#), [ST\\_M](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 7.4.17 ST\_IsClosed

**ST\_IsClosed** — Devuelve `TRUE` si los puntos de inicio y final de una `LINESTRING` son coincidentes. Para superficies poliedricas si son cerradas (volumetricas).

#### Synopsis

boolean **ST\_IsClosed**(geometry g);

#### Descripción

Devuelve `TRUE` si los puntos de inicio y final de una `LINESTRING` son coincidentes. Para superficies poliédricas , te dice si las superficies son áreas (abiertas) o si son volumétricas (cerradas).



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 7.1.5, 9.3.3



#### Note

SQL-MM define que el resultado de `ST_IsClosed(NULL)` debe ser 0, mientras que PostGIS devuelve `NULL`.



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



This method supports Circular Strings and Curves.

Mejora: 2.0.0 se introdujo soporte de superficies poliédricas.



This function supports Polyhedral surfaces.

#### Ejemplos con lineas y puntos

```
postgis=# SELECT ST_IsClosed('LINESTRING(0 0, 1 1)::geometry');
st_isclosed

f
(1 row)

postgis=# SELECT ST_IsClosed('LINESTRING(0 0, 0 1, 1 1, 0 0)::geometry');
st_isclosed

t
(1 row)

postgis=# SELECT ST_IsClosed('MULTILINESTRING((0 0, 0 1, 1 1, 0 0),(0 0, 1 1))::geometry');
st_isclosed

f
(1 row)

postgis=# SELECT ST_IsClosed('POINT(0 0)::geometry');
st_isclosed

t
```

```
(1 row)

postgis=# SELECT ST_IsClosed('MULTIPOINT((0 0), (1 1))'::geometry);
st_isclosed

t
(1 row)
```

### Ejemplos con superficies Poliédricas

```
-- A cube --
SELECT ST_IsClosed(ST_GeomFromEWKT('POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 ↵
1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0) ↵
),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1) ↵
))')));

st_isclosed

t

-- Same as cube but missing a side --
SELECT ST_IsClosed(ST_GeomFromEWKT('POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 ↵
0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0) ↵
),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)))')));

st_isclosed

f
```

### Ver también

[ST\\_IsRing](#)

## 7.4.18 ST\_IsCollection

**ST\_IsCollection** — Devuelve True si la Geometría es una colección vacía, polígono vacío, punto vacío etc.

### Synopsis

boolean **ST\_IsCollection**(geometry g);

### Descripción

Devuelve TRUE si la geometría del argumento es:

- GEOMETRYCOLLECTION



- MULTI{POINT,POLYGON,LINestring,CURVE,SURFACE}
- COMPOUNDCURVE

**Note**

Esta función analiza el tipo de geometría. Esto significa que devolverá `TRUE` en colecciones que estén vacías o que contengan un único elemento.



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



This method supports Circular Strings and Curves.

**Ejemplos**

```
postgis=# SELECT ST_IsCollection('LINESTRING(0 0, 1 1)::geometry');
st_iscollection

f
(1 row)

postgis=# SELECT ST_IsCollection('MULTIPOINT EMPTY)::geometry');
st_iscollection

t
(1 row)

postgis=# SELECT ST_IsCollection('MULTIPOINT((0 0))::geometry');
st_iscollection

t
(1 row)

postgis=# SELECT ST_IsCollection('MULTIPOINT((0 0), (42 42))::geometry');
st_iscollection

t
(1 row)

postgis=# SELECT ST_IsCollection('GEOMETRYCOLLECTION(POINT(0 0))::geometry');
st_iscollection

t
(1 row)
```

**Ver también**

[ST\\_NumGeometries](#)

**7.4.19 ST\_IsEmpty**

`ST_IsEmpty` — Tests if a geometry is empty.

**Synopsis**

boolean **ST\_IsEmpty**(geometry geomA);

## Descripción

Devuelve True si la Geometría es una geometría vacía. Si es cierto, entonces esta Geometría representa una colección de geometrías vacías, polígonos vacíos, puntos vacíos, etc.



### Note

SQL-MM define que el resultado de ST\_IsEmpty(NULL) debe ser 0, mientras que PostGIS devuelve NULL.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



This method implements the SQL/MM specification. SQL-MM 3: 5.1.7



This method supports Circular Strings and Curves.



### Warning

Cambiado: 2.0.0 En las versiones anteriores de PostGIS ST\_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') estaba permitido. Esto no esta permitido ahora en PostGIS 2.0.0 para ajustarse mejor a las normas SQL/MM.

## Ejemplos

```
SELECT ST_IsEmpty(ST_GeomFromText('GEOMETRYCOLLECTION EMPTY'));
st_isempty

t
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON EMPTY'));
st_isempty

t
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))'));
st_isempty

f
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))')) = false;
?column?

t
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('CIRCULARSTRING EMPTY'));
st_isempty

t
(1 row)
```

### 7.4.20 ST\_IsPolygonCCW

**ST\_IsPolygonCCW** — Devuelve true si todos los aros exteriores están orientados hacia la izquierda y todos los aros interiores están orientados hacia la derecha.

#### Synopsis

boolean **ST\_IsPolygonCCW** ( geometry geom );

#### Descripción

Devuelve true si todos los componentes poligonales de la geometría de entrada utilizan una orientación contraria a las manecillas del reloj para su aro exterior y una dirección en el sentido de las manecillas del reloj para todos los anillos interiores.

Devuelve true si la geometría no tiene componentes poligonales.



#### Note

Cadenas de líneas cerradas no se consideran componentes poligonales, por lo que aún obtendrá como devolución verdadero por pasar una sola cadena de líneas cerrada sin importar su orientación.



#### Note

Si una geometría poligonal no utiliza la orientación inversa para los anillos interiores (es decir, si uno o más anillos interiores están orientados en la misma dirección que un anillo exterior), ambos **ST\_IsPolygonCW** y **ST\_IsPolygonCCW** devolverán false.

Disponibilidad: 2.2.0



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



This function supports M coordinates.

#### Ver también

**ST\_ForcePolygonCW** , **ST\_ForcePolygonCCW** , **ST\_IsPolygonCW**

### 7.4.21 ST\_IsPolygonCW

**ST\_IsPolygonCW** — Devuelve true si todos los aros exteriores están orientados hacia la derecha y todos los aros interiores están orientados en sentido contrario a las agujas del reloj.

#### Synopsis

boolean **ST\_IsPolygonCW** ( geometry geom );

## Descripción

Devuelve true si todos los componentes poligonales de la geometría de entrada utilizan una orientación horaria para su aro exterior y una dirección contraria a las manecillas del reloj para todos los anillos interiores.

Devuelve true si la geometría no tiene componentes poligonales.



### Note

Cadenas de líneas cerradas no se consideran componentes poligonales, por lo que aún obtendrá como devolución verdadero por pasar una sola cadena de líneas cerrada sin importar su orientación.



### Note

Si una geometría poligonal no utiliza la orientación inversa para los anillos interiores (es decir, si uno o más anillos interiores están orientados en la misma dirección que un anillo exterior), ambos `ST_IsPolygonCW` y `ST_IsPolygonCCW` devolverán false.

Disponibilidad: 2.2.0



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



This function supports M coordinates.

## Ver también

[ST\\_ForcePolygonCW](#) , [ST\\_ForcePolygonCCW](#) , [ST\\_IsPolygonCW](#)

## 7.4.22 ST\_IsRing

`ST_IsRing` — Tests if a LineString is closed and simple.

## Synopsis

boolean **ST\_IsRing**(geometry g);

## Descripción

Devuelve TRUE si esta `LINESTRING` es **ST\_IsClosed** (`ST_StartPoint((g)) ~= ST_Endpoint((g))`) y **ST\_IsSimple** (no se intersecta con ella misma).



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). 2.1.5.1



This method implements the SQL/MM specification. SQL-MM 3: 7.1.6



### Note

SQL-MM define que el resultado de `ST_IsRing(NULL)` debe ser 0, mientras que PostGIS devuelve NULL.

## Ejemplos

```
SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom)
FROM (SELECT 'LINESTRING(0 0, 0 1, 1 1, 1 0, 0 0)::geometry AS geom) AS foo;
 st_isring | st_isclosed | st_issimple
-----+-----+-----
t | t | t
(1 row)

SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom)
FROM (SELECT 'LINESTRING(0 0, 0 1, 1 0, 1 1, 0 0)::geometry AS geom) AS foo;
 st_isring | st_isclosed | st_issimple
-----+-----+-----
f | t | f
(1 row)
```

## Ver también

[ST\\_IsClosed](#), [ST\\_IsSimple](#), [ST\\_StartPoint](#), [ST\\_EndPoint](#)

### 7.4.23 ST\_IsSimple

**ST\_IsSimple** — Devuelve (TRUE) si la geometría no tiene puntos geométricos anómalos, como auto intersecciones o tangencias.

## Synopsis

boolean **ST\_IsSimple**(geometry geomA);

## Descripción

Devuelve TRUE si la geometría no tiene puntos geométricos anómalos, como auto intersecciones o tangencias. Para mas información sobre la definición del OGC de simplicidad y validez geométrica, visita el enlace ["Ensuring OpenGIS compliancy of geometries"](#)



### Note

SQL-MM define que el resultado de `ST_IsSimple(NULL)` debe ser 0, mientras que PostGIS devuelve NULL.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



This method implements the SQL/MM specification. SQL-MM 3: 5.1.8



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

## Ejemplos

```
SELECT ST_IsSimple(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))'));
 st_issimple

t
(1 row)
```

```
SELECT ST_IsSimple(ST_GeomFromText('LINESTRING(1 1,2 2,2 3.5,1 3,1 2,2 1)'));
st_issimple

f
(1 row)
```

**Ver también**

[ST\\_IsValid](#)

### 7.4.24 ST\_M

**ST\_M** — Returns the M coordinate of a Point.

#### Synopsis

float **ST\_M**(geometry a\_point);

#### Descripción

Devuelve la coordenada M del punto, o NULL si no seta disponible. La entrada debe ser un punto.



#### Note

Esto no es (todavía) parte de la especificación OGC, pero esta incluida aquí para completar la lista de extracción de coordenadas de un punto.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification.



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

#### Ejemplos

```
SELECT ST_M(ST_GeomFromEWKT('POINT(1 2 3 4)'));
st_m

4
(1 row)
```

**Ver también**

[ST\\_GeomFromEWKT](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_Z](#)

### 7.4.25 ST\_MemSize

**ST\_MemSize** — Devuelve el tipo de geometría del valor de ST\_Geometry.

Synopsis

integer **ST\_NRings**(geometry geomA);

Descripción

Devuelve el tipo de geometría del valor de ST\_Geometry.

This complements the PostgreSQL built-in [database object functions](#) `pg_column_size`, `pg_size_pretty`, `pg_relation_size`, `pg_total_relation_size`.



**Note**  
`pg_relation_size` which gives the byte size of a table may return byte size lower than `ST_MemSize`. This is because `pg_relation_size` does not add toasted table contribution and large geometries are stored in TOAST tables.  
`pg_total_relation_size` - includes, the table, the toasted tables, and the indexes.  
`pg_column_size` returns how much space a geometry would take in a column considering compression, so may be lower than `ST_MemSize`

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Changed: 2.2.0 name changed to `ST_MemSize` to follow naming convention.

Ejemplos

```
--Return how much byte space Boston takes up in our Mass data set
SELECT pg_size_pretty(SUM(ST_MemSize(geom))) as totgeomsum,
pg_size_pretty(SUM(CASE WHEN town = 'BOSTON' THEN ST_MemSize(geom) ELSE 0 END)) As bossum,
CAST(SUM(CASE WHEN town = 'BOSTON' THEN ST_MemSize(geom) ELSE 0 END)*1.00 /
 SUM(ST_MemSize(geom))*100 As numeric(10,2)) As perbos
FROM towns;

totgeomsum bossum perbos

1522 kB 30 kB 1.99

SELECT ST_MemSize(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'));

73

--What percentage of our table is taken up by just the geometry
SELECT pg_total_relation_size('public.neighborhoods') As fulltable_size, sum(ST_MemSize(geom)) As geomsizes,
sum(ST_MemSize(geom))*1.00/pg_total_relation_size('public.neighborhoods')*100 As pergeom
FROM neighborhoods;
fulltable_size geomsizes pergeom

262144 96238 36.71188354492187500000
```

### 7.4.26 ST\_NDims

ST\_NDims — Devuelve la dimensión de las coordenadas del valor de ST\_Geometry.

#### Synopsis

integer **ST\_NDims**(geometry g1);

#### Descripción

Devuelve la dimension de las coordenadas de la geometría. PostGIS soporta 2 - (x,y), 3 - (x,y,z) o 2D con medidas - x,y,m y 4 -3D con medidas en el espacio x,y,z,m.



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

#### Ejemplos

```
SELECT ST_NDims(ST_GeomFromText('POINT(1 1)')) As d2point,
 ST_NDims(ST_GeomFromEWKT('POINT(1 1 2)')) As d3point,
 ST_NDims(ST_GeomFromEWKT('POINTM(1 1 0.5)')) As d2pointm;
```

| d2point | d3point | d2pointm |
|---------|---------|----------|
| 2       | 3       | 3        |

#### Ver también

[ST\\_CoordDim](#), [ST\\_Dimension](#), [ST\\_GeomFromEWKT](#)

### 7.4.27 ST\_NPoints

ST\_NPoints — Devuelve el numero de puntos (vértices) en la geometría.

#### Synopsis

integer **ST\_NPoints**(geometry g1);

#### Descripción

Devuelve el numero de puntos en la geometría. Funciona con todas las geometrías.

Mejora: 2.0.0 se introdujo soporte de superficies poliédricas.



#### Note

Anterior a 1.3.4, esta función daba errores si se utilizaba con geometrías que contenían CURVES. Esto se corrigió en 1.3.4+



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



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



## Ejemplos

```
SELECT ST_NPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
--result
4

--Polygon in 3D space
SELECT ST_NPoints(ST_GeomFromEWKT('LINESTRING(77.29 29.07 1,77.42 29.26 0,77.27 29.31 -1,77.29 29.07 3)'));
--result
4
```

## Ver también

[ST\\_NumPoints](#)

### 7.4.28 ST\_NRings

ST\_NRings — Devuelva el número de anillos interiores de una geometría poligonal.

## Synopsis

integer **ST\_NRings**(geometry geomA);

## Descripción

Si la geometría es un polígono o un multi-polígono devuelve el número de anillos. Al contrario que NumInteriorRings, cuenta el anillo exterior también.



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



This method supports Circular Strings and Curves.

## Ejemplos

```
SELECT ST_NRings(geom) As Nrings, ST_NumInteriorRings(geom) As ninterrings
FROM (SELECT ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))') As geom) As foo;
```

| nrings | ninterrings |
|--------|-------------|
| 1      | 0           |

(1 row)

## Ver también

[ST\\_NumInteriorRings](#)

### 7.4.29 ST\_NumGeometries

ST\_NumGeometries — Devuelve el número de puntos en la geometría. Funciona con todas las geometrías.

**Synopsis**

integer **ST\_NumGeometries**(geometry geom);

**Descripción**

Devuelve el numero de geometrías. Si la geometría es una GEOMETRYCOLLECTION (o MULTI\*) devuelve el numero de geometrías, para geometrías simples devuelve 1, si no devuelve NULL.

Mejorado: 2.0.0 se introdujo soporte para superficies poliédricas, Triangulos y TIN.

Cambiado: 2.0.0 En versiones anteriores esto devolvería NULL si la geometría no era de tipo collection/MULTI. 2.0.0+ devuelve 1 para geometrías simples, por ejemplo, POLYGON, LINESTRING, POINT.



This method implements the SQL/MM specification. SQL-MM 3: 9.1.4



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



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**Ejemplos**

```
--Prior versions would have returned NULL for this -- in 2.0.0 this returns 1
SELECT ST_NumGeometries(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
--result
1

--Geometry Collection Example - multis count as one geom in a collection
SELECT ST_NumGeometries(ST_GeomFromEWKT('GEOMETRYCOLLECTION(MULTIPOINT((-2 3),(-2 2)),
LINESTRING(5 5 ,10 10),
POLYGON((-7 4.2,-7.1 5,-7.1 4.3,-7 4.2)))'));
--result
3
```

**Ver también**

[ST\\_GeometryN](#), [ST\\_Multi](#)

**7.4.30 ST\_NumInteriorRings**

**ST\_NumInteriorRings** — Devuelva el número de anillos interiores de una geometría poligonal.

**Synopsis**

integer **ST\_NumInteriorRings**(geometry a\_polygon);

**Descripción**

Devuelve el número de anillos interiores de una geometría poligonal. Devuelve NULL si la geometría no es un polígono.



This method implements the SQL/MM specification. SQL-MM 3: 8.2.5

Cambiado: 2.0.0 - En versiones anteriores permitiría pasar un multipolígono, devolviendo el número de anillos interiores de primer polígono.

## Ejemplos

```
--If you have a regular polygon
SELECT gid, field1, field2, ST_NumInteriorRings(geom) AS numholes
FROM sometable;

--If you have multipolygons
--And you want to know the total number of interior rings in the MULTIPOLYGON
SELECT gid, field1, field2, SUM(ST_NumInteriorRings(geom)) AS numholes
FROM (SELECT gid, field1, field2, (ST_Dump(geom)).geom As geom
 FROM sometable) As foo
GROUP BY gid, field1,field2;
```

## Ver también

[ST\\_NumInteriorRing](#), [ST\\_PointN](#)

### 7.4.31 ST\_NumInteriorRing

**ST\_NumInteriorRing** — Devuelve el número de anillos interiores de un polígono en la geometría. Sinónimo de **ST\_NumInteriorRings**.

#### Synopsis

integer **ST\_NumInteriorRing**(geometry a\_polygon);

## Ver también

[ST\\_NumInteriorRings](#), [ST\\_PointN](#)

### 7.4.32 ST\_NumPatches

**ST\_NumPatches** — Devuelve el número de caras en una superficie poliédrica. Devolverá nulo para geometrías no poliédricas.

#### Synopsis

integer **ST\_NumPatches**(geometry g1);

#### Descripción

Devuelve el número de caras en una superficie poliédrica. Devolverá nulo para geometrías no poliédricas. Esto es un alias para **ST\_NumGeometries** para admitir nombres MM. Más rápido para usar **ST\_NumGeometries** si no te importa la convención MM.

Disponibilidad: 2.0.0



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



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5



This function supports Polyhedral surfaces.

## Ejemplos

```
SELECT ST_NumPatches(ST_GeomFromEWKT('POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 ←
0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0) ←
),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1) ←
))'));
--result
6
```

## Ver también

[ST\\_GeomFromEWKT](#), [ST\\_NumGeometries](#)

### 7.4.33 ST\_NumPoints

**ST\_NumPoints** — Devuelve el número de puntos en un valor **ST\_LineString** o **ST\_CircularString**.

## Synopsis

integer **ST\_NumPoints**(geometry g1);

## Descripción

Devuelve el número de puntos en un valor **ST\_LineString** o **ST\_CircularString**. Antes de 1.4 sólo funcionaba con cadenas de línea como el estado de especificaciones. A partir de 1.4, esto es un alias para **ST\_NPoints** que devuelve el número de vértices para no sólo las cadenas de línea. Considere el uso de **ST\_NPoints** en su lugar, que es multiuso y funciona con muchos tipos de geometría.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 7.2.4

## Ejemplos

```
SELECT ST_NumPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 ←
29.07)'));
--result
4
```

## Ver también

[ST\\_NPoints](#)

### 7.4.34 ST\_PatchN

**ST\_PatchN** — Devuelve el tipo de geometría del valor de **ST\_Geometry**.

Synopsis

geometry **ST\_PatchN**(geometry geomA, integer n);

Descripción

Devuelve la 1 geometría de base n-ésima (cara) si la geometría es un POLYHEDRALSURFACE, POLYHEDRALSURFACEM. De lo contrario, devuelve NULL. Esto devuelve la misma respuesta que ST\_GeometryN para las superficies de poliedros. Usar ST\_GeometryN es más rápido.



**Note**  
El índice está basado en 1.



**Note**  
Si desea extraer todas las geometrías, de una geometría, ST\_Dump es más eficiente.

Disponibilidad: 2.0.0

- ✓ This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5
- ✓ This function supports 3d and will not drop the z-index.
- ✓ This function supports Polyhedral surfaces.

Ejemplos

```
--Extract the 2nd face of the polyhedral surface
SELECT ST_AsEWKT(ST_PatchN(geom, 2)) As geomewkt
FROM (
VALUES (ST_GeomFromEWKT('POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
 ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
 ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
 ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))'))) ←
 As foo(geom);

 geomewkt
-----+-----
POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))
```

Ver también

[ST\\_AsEWKT](#), [ST\\_GeomFromEWKT](#), [ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

7.4.35 ST\_PointN

ST\_PointN — Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString.

## Synopsis

geometry **ST\_PointN**(geometry a\_linestring, integer n);

## Descripción

Devuelve el punto *n*-ésimo en una sola cadena de línea o cadena de línea circular en la geometría. Los valores negativos se contabilizan hacia atrás desde el final de la cadena de línea, por lo que -1 es el último punto. Devuelve NULL si no hay cadena de línea en la geometría.



### Note

El índice se basa en 1 como para las especificaciones OGC desde la versión 0.8.0. La indexación hacia atrás (índice negativo) no se encuentra en versiones anteriores de OGC implementado esto como basado en 0 en su lugar.



### Note

Si desea obtener el punto *n*-ésimo de cada cadena de línea en una multiple cadena de línea, utilícelo en conjunción con ST\_Dump



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 7.2.5, 7.3.5



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



This method supports Circular Strings and Curves.



### Note

Cambiado: 2.0.0 ya no funciona con una sola geometría multilinestrings. En versiones antiguas de PostGIS -- una sola línea MultiLineString trabajaría felizmente con esta función y regresaría el punto de inicio. En 2.0.0 sólo devuelve NULL como cualquier otro MultiLineString.

Cambiado: 2.3.0: indexación negativa disponible (-1 es el último punto)

## Ejemplos

```
-- Extract all POINTs from a LINESTRING
SELECT ST_AsText(
 ST_PointN(
 column1,
 generate_series(1, ST_NPoints(column1))
))
FROM (VALUES ('LINESTRING(0 0, 1 1, 2 2)::geometry')) AS foo;

 st_astext

POINT(0 0)
POINT(1 1)
POINT(2 2)
(3 rows)

--Example circular string
```

```
SELECT ST_AsText(ST_PointN(ST_GeomFromText('CIRCULARSTRING(1 2, 3 2, 1 2)'), 2));

st_astext

POINT(3 2)
(1 row)

SELECT ST_AsText(f)
FROM ST_GeomFromText('LINESTRING(0 0 0, 1 1 1, 2 2 2)') AS g
,ST_PointN(g, -2) AS f; -- 1 based index

st_astext

POINT Z (1 1 1)
(1 row)
```

**Ver también**

[ST\\_NPoints](#)

### 7.4.36 ST\_Points

**ST\_Points** — Devuelve un MultiPoint que contiene todas las coordenadas de una geometría.

#### Synopsis

geometry **ST\_Points**( geometry geom );

#### Descripción

Returns a MultiPoint containing all the coordinates of a geometry. Duplicate points are preserved, including the start and end points of ring geometries. (If desired, duplicate points can be removed by calling [ST\\_RemoveRepeatedPoints](#) on the result).

To obtain information about the position of each coordinate in the parent geometry use [ST\\_NumPoints](#).

M and Z coordinates are preserved if present.



This method supports Circular Strings and Curves.



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

Disponibilidad: 2.3.0

#### Ejemplos

```
SELECT ST_AsText(ST_Points('POLYGON Z ((30 10 4,10 30 5,40 40 6, 30 10))'));

--result
MULTIPOINT Z ((30 10 4),(10 30 5),(40 40 6),(30 10 4))
```

**Ver también**

[ST\\_RemoveRepeatedPoints](#), [ST\\_PointN](#)

### 7.4.37 ST\_StartPoint

ST\_StartPoint — Returns the first point of a LineString.

#### Synopsis

geometry **ST\_StartPoint**(geometry geomA);

#### Descripción

Devuelve el primer punto de una geometría `LINESTRING` o `CIRCULARLINESTRING` como un `POINT` o `NULL` si el parámetro de entrada no es un `LINESTRING` o `CIRCULARLINESTRING`.



This method implements the SQL/MM specification. SQL-MM 3: 7.1.3



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



This method supports Circular Strings and Curves.

#### Note



Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString.

Cambiado: 2.0.0 ya no funciona con multilinestrings de geometrías simples. En versiones anteriores de PostGIS -- una linea simple multilinestring funciona sin problemas con esta función y devuelve el punto inicial. En la version 2.0.0 simplemente devuelve NULL como con cualquier multilinestring. La antigua version era una función sin documentar, pero la gente que asumía que tenía sus datos almacenados en `LINESTRING` pueden experimentar este comportamiento ahora de resultado `NULL` en la version 2.0.

#### Ejemplos

Start point of a LineString

```
SELECT ST_AsText(ST_StartPoint('LINESTRING(0 1, 0 2)::geometry'));
 st_astext

POINT(0 1)
```

Start point of a non-LineString is NULL

```
SELECT ST_StartPoint('POINT(0 1)::geometry') IS NULL AS is_null;
 is_null

t
```

Start point of a 3D LineString

```
SELECT ST_AsEWKT(ST_StartPoint('LINESTRING(0 1 1, 0 2 2)::geometry'));
 st_asewkt

POINT(0 1 1)
```

Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString.

```
SELECT ST_AsText(ST_StartPoint('CIRCULARSTRING(5 2,-3 1.999999, -2 1, -4 2, 6 3)::geometry' ←
));
 st_astext

POINT(5 2)
```



Ver también

[ST\\_EndPoint](#), [ST\\_PointN](#)

### 7.4.38 ST\_Summary

ST\_Summary — Devuelve un resumen de texto del contenido de la geometría.

#### Synopsis

text **ST\_Summary**(geometry g);  
text **ST\_Summary**(geography g);

#### Descripción

Devuelve un resumen de texto del contenido de la geometría.

Las banderas que se muestran entre corchetes después del tipo de geometría tienen el siguiente significado:

- M: tiene ordenada M
- Z: tiene ordenada Z
- B: Tiene un cuadro de delimitación en caché
- G: es geodésico (geography)
- S: tiene un sistema de referencia espacial



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Disponibilidad: 1.2.2

Mejorado: 2.0.0 agregó soporte para geography

Mejorada: 2.1.0 Indicador S para señalar si tiene un sistema de referencia espacial conocido

Mejorado: 2.2.0 agregó soporte para TIN y curvas

#### Ejemplos

```
=# SELECT ST_Summary(ST_GeomFromText('LINESTRING(0 0, 1 1)')) as geom,
 ST_Summary(ST_GeogFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) geog;
 geom | geog
-----+-----
 LineString[B] with 2 points | Polygon[BGS] with 1 rings
 | ring 0 has 5 points
 :
(1 row)

=# SELECT ST_Summary(ST_GeogFromText('LINESTRING(0 0 1, 1 1 1)')) As geog_line,
 ST_Summary(ST_GeomFromText('SRID=4326;POLYGON((0 0 1, 1 1 2, 1 2 3, 1 1 1, 0 0 1)) ←
 ')) As geom_poly;
```

```

;
 geog_line | geom_poly
-----+-----
LineString[ZBGS] with 2 points | Polygon[ZBS] with 1 rings
 | ring 0 has 5 points
 |
(1 row)
```

**Ver también**

[PostGIS\\_DropBBox](#), [PostGIS\\_AddBBox](#), [ST\\_Force3DM](#), [ST\\_Force3DZ](#), [ST\\_Force2D](#), [geography](#)  
[ST\\_IsValid](#), [ST\\_IsValid](#), [ST\\_IsValidReason](#), [ST\\_IsValidDetail](#)

**7.4.39 ST\_X**

**ST\_X** — Returns the X coordinate of a Point.

**Synopsis**

float **ST\_X**(geometry a\_point);

**Descripción**

Devuelve la coordenada X del punto, o NULL si no está disponible. La entrada debe ser un punto.



**Note**

To get the minimum and maximum X value of geometry coordinates use the functions [ST\\_XMin](#) and [ST\\_XMax](#).



This method implements the SQL/MM specification. SQL-MM 3: 6.1.3



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

**Ejemplos**

```
SELECT ST_X(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_x

 1
(1 row)

SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
 st_y

 1.5
(1 row)
```

**Ver también**

[ST\\_Centroid](#), [ST\\_GeomFromEWKT](#), [ST\\_M](#), [ST\\_XMax](#), [ST\\_XMin](#), [ST\\_Y](#), [ST\\_Z](#)

**7.4.40 ST\_Y**

**ST\_Y** — Returns the Y coordinate of a Point.

**Synopsis**

```
float ST_Y(geometry a_point);
```

**Descripción**

Devuelve la coordenada Y del punto, o NULL si no está disponible. La entrada debe ser un punto.

**Note**

To get the minimum and maximum Y value of geometry coordinates use the functions [ST\\_YMin](#) and [ST\\_YMax](#).



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 6.1.4



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

**Ejemplos**

```
SELECT ST_Y(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_y

 2
(1 row)

SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
 st_y

 1.5
(1 row)
```

**Ver también**

[ST\\_Centroid](#), [ST\\_GeomFromEWKT](#), [ST\\_M](#), [ST\\_X](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_Z](#)

**7.4.41 ST\_Z**

**ST\_Z** — Returns the Z coordinate of a Point.

**Synopsis**

float **ST\_Z**(geometry a\_point);

**Descripción**

Devuelve la coordenada Z del punto, o NULL si no está disponible. La entrada debe ser un punto.

**Note**

To get the minimum and maximum Z value of geometry coordinates use the functions **ST\_ZMin** and **ST\_ZMax**.



This method implements the SQL/MM specification.



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

**Ejemplos**

```
SELECT ST_Z(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_z

 3
(1 row)
```

**Ver también**

**ST\_GeomFromEWKT**, **ST\_M**, **ST\_X**, **ST\_Y**, **ST\_ZMax**, **ST\_ZMin**

**7.4.42 ST\_Zmflag**

**ST\_Zmflag** — Devuelve la dimensión de las coordenadas del valor de **ST\_Geometry**.

**Synopsis**

smallint **ST\_Zmflag**(geometry geomA);

**Descripción**

Devuelve la dimensión de las coordenadas del valor de **ST\_Geometry**.

Values are: 0 = 2D, 1 = 3D-M, 2 = 3D-Z, 3 = 4D.



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



This method supports Circular Strings and Curves.

## Ejemplos

```
SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRING(1 2, 3 4)'));
st_zmflag

0

SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRINGM(1 2 3, 3 4 3)'));
st_zmflag

1

SELECT ST_Zmflag(ST_GeomFromEWKT('CIRCULARSTRING(1 2 3, 3 4 3, 5 6 3)'));
st_zmflag

2

SELECT ST_Zmflag(ST_GeomFromEWKT('POINT(1 2 3 4)'));
st_zmflag

3
```

## Ver también

[ST\\_CoordDim](#), [ST\\_NDims](#), [ST\\_Dimension](#)

## 7.5 Editores de Geometría

### 7.5.1 ST\_AddPoint

**ST\_AddPoint** — Añade un punto a una cadena de línea.

#### Synopsis

geometry **ST\_AddPoint**(geometry linestring, geometry point);  
 geometry **ST\_AddPoint**(geometry linestring, geometry point, integer position = -1);

#### Descripción

Adds a point to a LineString before the index *position* (using a 0-based index). If the *position* parameter is omitted or is -1 the point is appended to the end of the LineString.

Disponibilidad: 1.1.0



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

## Ejemplos

Add a point to the end of a 3D line

```
SELECT ST_AsEWKT(ST_AddPoint('LINESTRING(0 0 1, 1 1 1)', ST_MakePoint(1, 2, 3)));

st_asewkt

LINESTRING(0 0 1,1 1 1,1 2 3)
```

Guarantee all lines in a table are closed by adding the start point of each line to the end of the line only for those that are not closed.

```
UPDATE sometable
SET geom = ST_AddPoint(geom, ST_StartPoint(geom))
FROM sometable
WHERE ST_IsClosed(geom) = false;
```

**Ver también**

[ST\\_RemovePoint](#), [ST\\_SetPoint](#)

## 7.5.2 ST\_CollectionExtract

**ST\_CollectionExtract** — Given a geometry collection, returns a multi-geometry containing only elements of a specified type.

### Synopsis

```
geometry ST_CollectionExtract(geometry collection);
geometry ST_CollectionExtract(geometry collection, integer type);
```

### Descripción

Given a geometry collection, returns a homogeneous multi-geometry.

If the *type* is not specified, returns a multi-geometry containing only geometries of the highest dimension. So polygons are preferred over lines, which are preferred over points.

If the *type* is specified, returns a multi-geometry containing only that type. If there are no sub-geometries of the right type, an EMPTY geometry is returned. Only points, lines and polygons are supported. The type numbers are:

- 1 == POINT
- 2 == LINESTRING
- 3 == POLYGON

For atomic geometry inputs, the geometry is returned unchanged if the input type matches the requested type. Otherwise, the result is an EMPTY geometry of the specified type. If required, these can be converted to multi-geometries using [ST\\_Multi](#).



#### Warning

MultiPolygon results are not checked for validity. If the polygon components are adjacent or overlapping the result will be invalid. (For example, this can occur when applying this function to an [ST\\_Split](#) result.) This situation can be checked with [ST\\_IsValid](#) and repaired with [ST\\_MakeValid](#).

Disponibilidad: 1.5.0



#### Note

Prior to 1.5.3 this function returned atomic inputs unchanged, no matter type. In 1.5.3 non-matching single geometries returned a NULL result. In 2.0.0 non-matching single geometries return an EMPTY result of the requested type.

## Ejemplos

Extract highest-dimension type:

```
SELECT ST_AsText(ST_CollectionExtract(
 'GEOMETRYCOLLECTION(POINT(0 0), LINESTRING(1 1, 2 2))');
 st_astext

 MULTILINESTRING((1 1, 2 2))
```

Extract points (type 1 == POINT):

```
SELECT ST_AsText(ST_CollectionExtract(
 'GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(POINT(0 0)))',
 1));
 st_astext

 MULTIPOINT((0 0))
```

Extract lines (type 2 == LINESTRING):

```
SELECT ST_AsText(ST_CollectionExtract(
 'GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(LINESTRING(0 0, 1 1)),LINESTRING(2 2, 3 3))' ←
 ,
 2));
 st_astext

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

Ver también

[ST\\_CollectionHomogenize](#), [ST\\_Multi](#), [ST\\_IsValid](#), [ST\\_MakeValid](#)

### 7.5.3 ST\_CollectionHomogenize

**ST\_CollectionHomogenize** — Returns the simplest representation of a geometry collection.

#### Synopsis

geometry **ST\_CollectionHomogenize**(geometry collection);

#### Descripción

Given a geometry collection, returns the "simplest" representation of the contents.

- Homogeneous (uniform) collections are returned as the appropriate multi-geometry.
- Heterogeneous (mixed) collections are flattened into a single GeometryCollection.
- Collections containing a single atomic element are returned as that element.
- Atomic geometries are returned unchanged. If required, these can be converted to a multi-geometry using [ST\\_Multi](#).



#### Warning

This function does not ensure that the result is valid. In particular, a collection containing adjacent or overlapping Polygons will create an invalid MultiPolygon. This situation can be checked with [ST\\_IsValid](#) and repaired with [ST\\_MakeValid](#).

Disponibilidad: 2.0.0

## Ejemplos

### Single-element collection converted to an atomic geometry

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0))'));

st_astext

POINT(0 0)
```

### Nested single-element collection converted to an atomic geometry:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(MULTIPOINT((0 0)))'));

st_astext

POINT(0 0)
```

### Collection converted to a multi-geometry:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0),POINT(1 1))'));

st_astext

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

### Nested heterogeneous collection flattened to a GeometryCollection:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0), GEOMETRYCOLLECTION ↵
(LINESTRING(1 1, 2 2)))'));

st_astext

GEOMETRYCOLLECTION(POINT(0 0),LINESTRING(1 1,2 2))
```

### Collection of Polygons converted to an (invalid) MultiPolygon:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION (POLYGON ((10 50, 50 50, 50 ↵
10, 10 10, 10 50)), POLYGON ((90 50, 90 10, 50 10, 50 50, 90 50)))'));

st_astext

MULTIPOLYGON(((10 50,50 50,50 10,10 10,10 50)),((90 50,90 10,50 10,50 50,90 50)))
```

## Ver también

[ST\\_CollectionExtract](#), [ST\\_Multi](#), [ST\\_IsValid](#), [ST\\_MakeValid](#)

## 7.5.4 ST\_CurveToLine

**ST\_CurveToLine** — Converts a geometry containing curves to a linear geometry.

### Synopsis

geometry **ST\_CurveToLine**(geometry curveGeom, float tolerance, integer tolerance\_type, integer flags);



## Descripción

Converts a CIRCULAR STRING to regular LINESTRING or CURVEPOLYGON to POLYGON or MULTISURFACE to MULTIPOLYGON. Useful for outputting to devices that can't support CIRCULARSTRING geometry types

Converts a given geometry to a linear geometry. Each curved geometry or segment is converted into a linear approximation using the given `tolerance` and options (32 segments per quadrant and no options by default).

The `tolerance\_type` argument determines interpretation of the `tolerance` argument. It can take the following values:

- 0 (default): Tolerance is max segments per quadrant.
- 1: Tolerance is max-deviation of line from curve, in source units.
- 2: Tolerance is max-angle, in radians, between generating radii.

The `flags` argument is a bitfield. 0 by default. Supported bits are:

- 1: Symmetric (orientation independent) output.
- 2: Retain angle, avoids reducing angles (segment lengths) when producing symmetric output. Has no effect when Symmetric flag is off.

Availability: 1.3.0

Enhanced: 2.4.0 added support for max-deviation and max-angle tolerance, and for symmetric output.

Enhanced: 3.0.0 implemented a minimum number of segments per linearized arc to prevent topological collapse.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 7.1.7



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



This method supports Circular Strings and Curves.

## Ejemplos

```
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)')));

--Result --
LINESTRING(220268 150415,220269.95064912 150416.539364228,220271.823415575 150418.17258804,220273.613787707 150419.895736857,
220275.317452352 150421.704659462,220276.930305234 150423.594998003,220278.448460847 150425.562198489,
220279.868261823 150427.60152176,220281.186287736 150429.708054909,220282.399363347 150431.876723113,
220283.50456625 150434.10230186,220284.499233914 150436.379429536,220285.380970099 150438.702620341,220286.147650624 150441.066277505,
220286.797428488 150443.464706771,220287.328738321 150445.892130112,220287.740300149 150448.342699654,
220288.031122486 150450.810511759,220288.200504713 150453.289621251,220288.248038775 150455.77405574,
220288.173610157 150458.257830005,220287.977398166 150460.734960415,220287.659875492 150463.199479347,
220287.221807076 150465.64544956,220286.664248262 150468.066978495,220285.988542259 150470.458232479,220285.196316903 150472.81345077,
220284.289480732 150475.126959442,220283.270218395 150477.39318505,220282.140985384 150479.606668057,
```

```

220280.90450212 150481.762075989,220279.5637474 150483.85421628,220278.12195122 ↵
150485.87804878,
220276.582586992 150487.828697901,220274.949363179 150489.701464356,220273.226214362 ↵
150491.491836488,
220271.417291757 150493.195501133,220269.526953216 150494.808354014,220267.559752731 ↵
150496.326509628,
220265.520429459 150497.746310603,220263.41389631 150499.064336517,220261.245228106 ↵
150500.277412127,
220259.019649359 150501.38261503,220256.742521683 150502.377282695,220254.419330878 ↵
150503.259018879,
220252.055673714 150504.025699404,220249.657244448 150504.675477269,220247.229821107 ↵
150505.206787101,
220244.779251566 150505.61834893,220242.311439461 150505.909171266,220239.832329968 ↵
150506.078553494,
220237.347895479 150506.126087555,220234.864121215 150506.051658938,220232.386990804 ↵
150505.855446946,
220229.922471872 150505.537924272,220227.47650166 150505.099855856,220225.054972724 ↵
150504.542297043,
220222.663718741 150503.86659104,220220.308500449 150503.074365683,
220217.994991777 150502.167529512,220215.72876617 150501.148267175,
220213.515283163 150500.019034164,220211.35987523 150498.7825509,
220209.267734939 150497.441796181,220207.243902439 150496,
220205.293253319 150494.460635772,220203.420486864 150492.82741196,220201.630114732 ↵
150491.104263143,
220199.926450087 150489.295340538,220198.313597205 150487.405001997,220196.795441592 ↵
150485.437801511,
220195.375640616 150483.39847824,220194.057614703 150481.291945091,220192.844539092 ↵
150479.123276887,220191.739336189 150476.89769814,
220190.744668525 150474.620570464,220189.86293234 150472.297379659,220189.096251815 ↵
150469.933722495,
220188.446473951 150467.535293229,220187.915164118 150465.107869888,220187.50360229 ↵
150462.657300346,
220187.212779953 150460.189488241,220187.043397726 150457.710378749,220186.995863664 ↵
150455.22594426,
220187.070292282 150452.742169995,220187.266504273 150450.265039585,220187.584026947 ↵
150447.800520653,
220188.022095363 150445.35455044,220188.579654177 150442.933021505,220189.25536018 ↵
150440.541767521,
220190.047585536 150438.18654923,220190.954421707 150435.873040558,220191.973684044 ↵
150433.60681495,
220193.102917055 150431.393331943,220194.339400319 150429.237924011,220195.680155039 ↵
150427.14578372,220197.12195122 150425.12195122,
220198.661315447 150423.171302099,220200.29453926 150421.298535644,220202.017688077 ↵
150419.508163512,220203.826610682 150417.804498867,
220205.716949223 150416.191645986,220207.684149708 150414.673490372,220209.72347298 ↵
150413.253689397,220211.830006129 150411.935663483,
220213.998674333 150410.722587873,220216.22425308 150409.61738497,220218.501380756 ↵
150408.622717305,220220.824571561 150407.740981121,
220223.188228725 150406.974300596,220225.586657991 150406.324522731,220227 150406)

--3d example
SELECT ST_AsEWKT(ST_CurveToLine(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 ↵
150505 2,220227 150406 3)')));
Output

LINESTRING(220268 150415 1,220269.95064912 150416.539364228 1.0181172856673,
220271.823415575 150418.17258804 1.03623457133459,220273.613787707 150419.895736857 ↵
1.05435185700189,...AD INFINITUM
220225.586657991 150406.324522731 1.32611114201132,220227 150406 3)

--use only 2 segments to approximate quarter circle

```

```

SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'),2));
st_astext

LINESTRING(220268 150415,220287.740300149 150448.342699654,220278.12195122 150485.87804878,
220244.779251566 150505.61834893,220207.243902439 150496,220187.50360229 150462.657300346,
220197.12195122 150425.12195122,220227 150406)

-- Ensure approximated line is no further than 20 units away from
-- original curve, and make the result direction-neutral
SELECT ST_AsText(ST_CurveToLine(
 'CIRCULARSTRING(0 0,100 -100,200 0)::geometry,
 20, -- Tolerance
 1, -- Above is max distance between curve and line
 1 -- Symmetric flag
));
st_astext

LINESTRING(0 0,50 -86.6025403784438,150 -86.6025403784439,200 -1.1331077795296e-13,200 0)

```

**Ver también**

[ST\\_LineToCurve](#)

### 7.5.5 ST\_Scroll

ST\_Scroll — Change start point of a closed LineString.

#### Synopsis

geometry **ST\_Scroll**(geometry linestring, geometry point);

#### Descripción

Changes the start/end point of a closed LineString to the given vertex *point*.

Availability: 3.2.0



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



This function supports M coordinates.

#### Ejemplos

Make e closed line start at its 3rd vertex

```

SELECT ST_AsEWKT(ST_Scroll('SRID=4326;LINESTRING(0 0 0 1, 10 0 2 0, 5 5 4 2,0 0 0 1)', 'POINT(5 5 4 2)'));
st_asewkt

SRID=4326;LINESTRING(5 5 4 2,0 0 0 1,10 0 2 0,5 5 4 2)

```

**Ver también**

[ST\\_Normalize](#)

### 7.5.6 ST\_FlipCoordinates

ST\_FlipCoordinates — Returns a version of a geometry with X and Y axis flipped.

#### Synopsis

geometry **ST\_FlipCoordinates**(geometry geom);

#### Descripción

Returns a version of the given geometry with X and Y axis flipped. Useful for fixing geometries which contain coordinates expressed as latitude/longitude (Y,X).

Disponibilidad: 2.0.0



This method supports Circular Strings and Curves.



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



This function supports M coordinates.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

#### Ejemplo

```
SELECT ST_AsEWKT(ST_FlipCoordinates(GeomFromEWKT('POINT(1 2)')));
 st_asewkt

POINT(2 1)
```

**Ver también**

[ST\\_SwapOrdinates](#)

### 7.5.7 ST\_Force2D

ST\_Force2D — Forzar las geometrías en un "modo de 2 dimensiones".

#### Synopsis

geometry **ST\_Force2D**(geometry geomA);

## Descripción

Forzar las geometrías en un "modo de 2 dimensiones" para que todas las representaciones de salida sólo tengan las coordenadas X e Y. Esto es útil para forzar la salida compatible con OGC (ya que OGC sólo especifica geometría 2D).

Mejorado: 2.0.0 soporte para superficies poliédricas fue introducida.

Cambiado: 2.1.0. Hasta la 2.0.x esto se llamaba ST\_Force\_2D.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



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

## Ejemplos

```
SELECT ST_AsEWKT(ST_Force2D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
 st_asewkt

CIRCULARSTRING(1 1,2 3,4 5,6 7,5 6)

SELECT ST_AsEWKT(ST_Force2D('POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))'));
 st_asewkt

POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))
```

## Ver también

[ST\\_Force3D](#)

## 7.5.8 ST\_Force3D

ST\_Force3D — Forzar las geometrías en modo XYZ. Este es un alias para ST\_Force3DZ.

## Synopsis

geometry **ST\_Force3D**(geometry geomA, float Zvalue = 0.0);

## Descripción

Forces the geometries into XYZ mode. This is an alias for ST\_Force3DZ. If a geometry has no Z component, then a *Zvalue* Z coordinate is tacked on.

Mejorado: 2.0.0 soporte para superficies poliédricas fue introducida.

Cambiado: 2.1.0. Hasta la 2.0.x esto se llamaba ST\_Force\_3D.

Changed: 3.1.0. Added support for supplying a non-zero Z value.



This function supports Polyhedral surfaces.



This method supports Circular Strings and Curves.



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

## Ejemplos

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
 st_asewkt

CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)

SELECT ST_AsEWKT(ST_Force3D('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));
 st_asewkt

POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

## Ver también

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3DZ](#)

## 7.5.9 ST\_Force3DZ

ST\_Force3DZ — Fuerza las geometrías en modo XYZ.

## Synopsis

geometry **ST\_Force3DZ**(geometry geomA, float Zvalue = 0.0);

## Descripción

Forces the geometries into XYZ mode. If a geometry has no Z component, then a *Zvalue* Z coordinate is tacked on.

Mejorado: 2.0.0 soporte para superficies poliédricas fue introducida.

Cambiado: 2.1.0. Hasta la 2.0.x esto se llamaba ST\_Force\_3DZ.

Changed: 3.1.0. Added support for supplying a non-zero Z value.



This function supports Polyhedral surfaces.



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



This method supports Circular Strings and Curves.

## Ejemplos

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3DZ(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
 st_asewkt

CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)

SELECT ST_AsEWKT(ST_Force3DZ('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));
 st_asewkt

POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

```

-----st_asewkt-----
POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))

```

#### Ver también

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3D](#)

### 7.5.10 ST\_Force3DM

ST\_Force3DM — Fuerza las geometrías en modo XYM.

#### Synopsis

geometry **ST\_Force3DM**(geometry geomA, float Mvalue = 0.0);

#### Descripción

Forces the geometries into XYM mode. If a geometry has no M component, then a *Mvalue* M coordinate is tacked on. If it has a Z component, then Z is removed

Cambiado: 2.1.0. Hasta la 2.0.x esto se llamaba ST\_Force\_3DM.

Changed: 3.1.0. Added support for supplying a non-zero M value.



This method supports Circular Strings and Curves.

#### Ejemplos

```

--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3DM(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
-----st_asewkt-----
CIRCULARSTRINGM(1 1 0,2 3 0,4 5 0,6 7 0,5 6 0)

SELECT ST_AsEWKT(ST_Force3DM('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1))'));
-----st_asewkt-----
POLYGONM((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))

```

#### Ver también

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3D](#), [ST\\_GeomFromEWKT](#)

### 7.5.11 ST\_Force4D

ST\_Force4D — Fuerza las geometrías en modo XYZM.

## Synopsis

geometry **ST\_Force4D**(geometry geomA, float Zvalue = 0.0, float Mvalue = 0.0);

## Descripción

Forces the geometries into XYZM mode. *Zvalue* and *Mvalue* is tacked on for missing Z and M dimensions, respectively.

Cambiado: 2.1.0. Hasta la 2.0.x esto se llamaba ST\_Force\_4D.

Changed: 3.1.0. Added support for supplying non-zero Z and M values.



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



This method supports Circular Strings and Curves.

## Ejemplos

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force4D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
```

|  | st_asewkt                                               |
|--|---------------------------------------------------------|
|  | CIRCULARSTRING(1 1 2 0,2 3 2 0,4 5 2 0,6 7 2 0,5 6 2 0) |

```
SELECT ST_AsEWKT(ST_Force4D('MULTILINESTRINGM((0 0 1,0 5 2,5 0 3,0 0 4),(1 1 1,3 1 1,1 3 1,1 1 1))'));
```

|  | st_asewkt                                                                            |
|--|--------------------------------------------------------------------------------------|
|  | MULTILINESTRING((0 0 0 1,0 5 0 2,5 0 0 3,0 0 0 4),(1 1 0 1,3 1 0 1,1 3 0 1,1 1 0 1)) |

## Ver también

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3D](#)

## 7.5.12 ST\_ForcePolygonCCW

ST\_ForcePolygonCCW — Orienta todos los aros exteriores en sentido contrario a las agujas del reloj y todos los aros interiores en sentido horario.

## Synopsis

geometry **ST\_ForcePolygonCCW** ( geometry geom );

## Descripción

Fuerza (Multi)polígonos a utilizar una orientación en sentido contrario a las manecillas del reloj para su anillo exterior, y una orientación en el sentido de las agujas del reloj para sus anillos interiores. Las geometrías no poligonales se devuelven sin cambios.

Availability: 2.4.0



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

#### Ver también

[ST\\_ForcePolygonCW](#) , [ST\\_IsPolygonCCW](#) , [ST\\_IsPolygonCW](#)

### 7.5.13 ST\_ForceCollection

ST\_ForceCollection — Convertir la geometría en una GEOMETRYCOLLECTION.

#### Synopsis

geometry **ST\_ForceCollection**(geometry geomA);

#### Descripción

Convierte la geometría en una GEOMETRYCOLLECTION. Esto es útil para simplificar la representación WKB.

Mejorado: 2.0.0 soporte para superficies poliédricas fue introducida.

Disponibilidad: 1.2.2, antes de 1.3.4 esta función se bloqueará con curvas. Esto se fija en 1.3.4 +

Cambiado: 2.1.0. Hasta la 2.0.x esto se llamaba ST\_Force\_Collection.

-  This function supports Polyhedral surfaces.
-  This function supports 3d and will not drop the z-index.
-  This method supports Circular Strings and Curves.

#### Ejemplos

```
SELECT ST_AsEWKT(ST_ForceCollection('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1))'));
 st_asewkt

GEOMETRYCOLLECTION(POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1)))

SELECT ST_AsText(ST_ForceCollection('CIRCULARSTRING(220227 150406,2220227 150407,220227 150406)'));
 st_astext

GEOMETRYCOLLECTION(CIRCULARSTRING(220227 150406,2220227 150407,220227 150406))
(1 row)

-- POLYHEDRAL example --
SELECT ST_AsEWKT(ST_ForceCollection('POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0))));
```

```
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))')')
----- st_asewkt -----
GEOMETRYCOLLECTION(
 POLYGON((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
 POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
 POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
 POLYGON((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
 POLYGON((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
 POLYGON((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
)
```

#### Ver también

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3D](#), [ST\\_GeomFromEWKT](#)

### 7.5.14 ST\_ForcePolygonCW

**ST\_ForcePolygonCW** — Orienta todos los anillos exteriores en el sentido de las agujas del reloj y todos los anillos interiores en sentido contrario a las agujas del reloj.

#### Synopsis

geometry **ST\_ForcePolygonCW** ( geometry geom );

#### Descripción

Fuerza (Multi)Polígonos a utilizar una orientación en el sentido de las agujas del reloj para su anillo exterior, y una orientación en sentido contrario a las agujas del reloj para sus anillos interiores. Las geometrías no poligonales se devuelven sin cambios.

Availability: 2.4.0



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



This function supports M coordinates.

#### Ver también

[ST\\_ForcePolygonCCW](#) , [ST\\_IsPolygonCCW](#) , [ST\\_IsPolygonCW](#)

### 7.5.15 ST\_ForceSFS

**ST\_ForceSFS** — Fuerza las geometrías para usar sólo los tipos de geometría SFS 1.1.

#### Synopsis

geometry **ST\_ForceSFS**(geometry geomA);  
 geometry **ST\_ForceSFS**(geometry geomA, text version);

**Descripción**

This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



This method supports Circular Strings and Curves.



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

**7.5.16 ST\_ForceRHR**

ST\_ForceRHR — Fuerza la orientación de los vértices en un polígono para seguir la regla de la mano derecha.

**Synopsis**

geometry **ST\_ForceRHR**(geometry g);

**Descripción**

Fuerce la orientación de los vértices en un polígono para seguir la regla de la mano derecha, en el cual, el área que está delimitada por el polígono está a la derecha del límite. En particular, el anillo exterior está orientado en el sentido de las agujas del reloj y el interior está orientado en sentido contrario a las agujas del reloj. Esta función es sinónimo de **ST\_ForcePolygonCW**

**Note**

La definición anterior de la regla de la derecha entra en conflicto con definiciones utilizadas en otros contextos. Para evitar la confusión, se recomienda utilizar ST\_ForcePolygonCW.

Mejorado: 2.0.0 soporte para superficies poliédricas fue introducida.



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



This function supports Polyhedral surfaces.

**Ejemplos**

```
SELECT ST_AsEWKT (
 ST_ForceRHR (
 'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))'
)
);
```

|  | st_asewkt                                                    |
|--|--------------------------------------------------------------|
|  | POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2)) |

(1 row)

**Ver también**

**ST\_ForcePolygonCCW** , **ST\_ForcePolygonCW** , **ST\_IsPolygonCCW** , **ST\_IsPolygonCW** , **ST\_BuildArea**, **ST\_Polygonize**, **ST\_Reverse**

### 7.5.17 ST\_ForceCurve

ST\_ForceCurve — Relanzar una geometría en su tipo curvo, si corresponde.

#### Synopsis

geometry **ST\_ForceCurve**(geometry g);

#### Descripción

Convierte una geometría en su representación curvada, si corresponde: las líneas se convierten en curvas compuestas, las multi-líneas se convierten en polígonos multicurvos se convierten en polígonos de curvas los multipolígonos se convierten en multisuperficies. Si la entrada de geometría es ya una representación curvada regresa igual que la entrada.

Disponibilidad: 2.2.0



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



This method supports Circular Strings and Curves.

#### Ejemplos

```
SELECT ST_AsText (
 ST_ForceCurve (
 'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))'::geometry
)
);
```

|         | st_astext                                                            |
|---------|----------------------------------------------------------------------|
|         | CURVEPOLYGON Z ((0 0 2,5 0 2,0 5 2,0 0 2),(1 1 2,1 3 2,3 1 2,1 1 2)) |
| (1 row) |                                                                      |

#### Ver también

[ST\\_LineToCurve](#)

### 7.5.18 ST\_LineToCurve

ST\_LineToCurve — Converts a linear geometry to a curved geometry.

#### Synopsis

geometry **ST\_LineToCurve**(geometry geomANoncircular);

#### Descripción

Converts plain LINestring/POLYGON to CIRCULAR STRINGs and Curved Polygons. Note much fewer points are needed to describe the curved equivalent.



#### Note

If the input LINestring/POLYGON is not curved enough to clearly represent a curve, the function will return the same input geometry.

Availability: 1.3.0

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.

Ejemplos

```
-- 2D Example
SELECT ST_AsText(ST_LineToCurve(foo.geom)) As curvedastext, ST_AsText(foo.geom) As
 non_curvedastext
FROM (SELECT ST_Buffer('POINT(1 3)::geometry', 3) As geom) As foo;
```

| curvedastext                                                                     | non_curvedastext                                                       |
|----------------------------------------------------------------------------------|------------------------------------------------------------------------|
| CURVEPOLYGON(CIRCULARSTRING(4 3,3.12132034355964 0.878679656440359,   POLYGON((4 | POLYGON((4                                                             |
| 3,3.94235584120969 2.41472903395162,3.77163859753386 1.85194970290473,           | 3,3.94235584120969 2.41472903395162,3.77163859753386 1.85194970290473, |
| 1 0,-1.12132034355965 5.12132034355963,4 3))                                     | 1 0,-1.12132034355965 5.12132034355963,4 3))                           |
| 1.33328930094119,3.12132034355964 0.878679656440359,                             | 1.33328930094119,3.12132034355964 0.878679656440359,                   |
|                                                                                  | 3.49440883690764                                                       |
|                                                                                  | 2.66671069905881                                                       |
|                                                                                  | 0.505591163092366,2.14805029                                           |
|                                                                                  | 0.228361402466141,                                                     |
|                                                                                  | 1.58527096604839                                                       |
|                                                                                  | 0.0576441587903094,1                                                   |
|                                                                                  | 0,                                                                     |
|                                                                                  | 0.414729033951621                                                      |
|                                                                                  | 0.0576441587903077,-0.1480502                                          |
|                                                                                  | 0.228361402466137,                                                     |
|                                                                                  | -0.666710699058802                                                     |
|                                                                                  | 0.505591163092361,-1.1213203                                           |
|                                                                                  | 0.878679656440353,                                                     |
|                                                                                  | -1.49440883690763                                                      |
|                                                                                  | 1.33328930094119,-1.77163859                                           |
|                                                                                  | 1.85194970290472                                                       |
|                                                                                  | --ETC--                                                                |
|                                                                                  | ,3.94235584120969                                                      |
|                                                                                  | 3.58527096604839,4                                                     |
|                                                                                  | 3))                                                                    |

```
--3D example
SELECT ST_AsText(ST_LineToCurve(geom)) As curved, ST_AsText(geom) AS not_curved
FROM (SELECT ST_Translate(ST_Force3D(ST_Boundary(ST_Buffer(ST_Point(1,3), 2,2))),0,0,3) AS
 geom) AS foo;
```

| curved                                               | not_curved                          |
|------------------------------------------------------|-------------------------------------|
| CIRCULARSTRING Z (3 3 3,-1 2.99999999999999 3,3 3 3) | LINESTRING Z (3 3 3,2.4142135623731 |
| 1.58578643762691 3,1 1 3,                            | -0.414213562373092 1.5857864376269  |
|                                                      | 3,-1 2.99999999999999 3,            |
|                                                      | -0.414213562373101 4.41421356237309 |
|                                                      | 3,                                  |
|                                                      | 0.999999999999991 5                 |
|                                                      | 3,2.41421356237309 4.4142135623731  |
|                                                      | 3,3 3 3)                            |

(1 row)

**Ver también**

[ST\\_CurveToLine](#)

### 7.5.19 ST\_Multi

**ST\_Multi** — Devuelve la geometría como una geometría MULTI\*.

#### Synopsis

geometry **ST\_Multi**(geometry geom);

#### Descripción

Returns the geometry as a MULTI\* geometry collection. If the geometry is already a collection, it is returned unchanged.

#### Ejemplos

```
SELECT ST_AsText(ST_Multi('POLYGON ((10 30, 30 30, 30 10, 10 10, 10 30))'));
 st_astext

MULTIPOLYGON(((10 30,30 30,30 10,10 10,10 30)))
```

**Ver también**

[ST\\_AsText](#)

### 7.5.20 ST\_LineExtend

**ST\_LineExtend** — Returns a line with the last and first segments extended the specified distance(s).

#### Synopsis

geometry **ST\_LineExtend**(geometry line, float distance\_forward, float distance\_backward=0.0);

#### Descripción

Returns a line with the last and first segments extended the specified distance(s). Distance of zero carries out no extension. Only non-negative distances are allowed. The first (and last) two distinct points in a line are used to determine the direction of projection, duplicate points are ignored.

Availability: 3.4.0

#### Example: Projected point at 100,000 meters and bearing 45 degrees

```
SELECT ST_AsText(ST_Project('POINT(0 0)::geography', 100000, radians(45.0)));

POINT(0.635231029125537 0.639472334729198)
```

**Ver también**[ST\\_LocateAlong](#), [ST\\_Project](#)**7.5.21 ST\_Normalize**

**ST\_Normalize** — Devuelve la geometría en su forma canónica.

**Synopsis**

geometry **ST\_Normalize**(geometry geom);

**Descripción**

Devuelve la geometría en su forma normalizada/canónica. Puede reordenar vértices en anillos poligonales, anillos en un polígono, elementos en un complejo de geometría múltiple.

Principalmente útil sólo para propósitos de prueba (comparando los resultados esperados y los obtenidos).

Disponibilidad: 2.3.0

**Ejemplos**

```

SELECT ST_AsText(ST_Normalize(ST_GeomFromText(
 'GEOMETRYCOLLECTION(
 POINT(2 3),
 MULTILINESTRING((0 0, 1 1),(2 2, 3 3)),
 POLYGON(
 (0 10,0 0,10 0,10 10,0 10),
 (4 2,2 2,2 4,4 4,4 2),
 (6 8,8 8,8 6,6 6,6 8)
)
) '
))) ;

```

st\_astext

---

```

GEOMETRYCOLLECTION(POLYGON((0 0,0 10,10 10,10 0,0 0),(6 6,8 6,8 8,6 8,6 6),(2 2,4 2,4 4,2 4,2 2)),MULTILINESTRING((2 2,3 3),(0 0,1 1)),POINT(2 3))
(1 row)

```

**Ver también**[ST\\_Equals](#),**7.5.22 ST\_Project**

**ST\_Project** — Returns a point projected from a start point by a distance and bearing (azimuth).

**Synopsis**

geometry **ST\_Project**(geometry g1, float distance, float azimuth);  
 geometry **ST\_Project**(geometry g1, geometry g2, float distance);  
 geography **ST\_Project**(geography g1, float distance, float azimuth);  
 geography **ST\_Project**(geography g1, geography g2, float distance);

## Descripción

Returns a point projected from a point along a geodesic using a given distance and azimuth (bearing). This is known as the direct geodesic problem.

The two-point version uses the path from the first to the second point to implicitly define the azimuth and uses the distance as before.

The distance is given in meters. Negative values are supported.

The azimuth (also known as heading or bearing) is given in radians. It is measured clockwise from true north.

- North is azimuth zero (0 degrees)
- East is azimuth  $\pi/2$  (90 degrees)
- South is azimuth  $\pi$  (180 degrees)
- West is azimuth  $3\pi/2$  (270 degrees)

Negative azimuth values and values greater than  $2\pi$  (360 degrees) are supported.

Disponibilidad: 2.0.0

Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth.

Enhanced: 3.4.0 Allow geometry arguments and two-point form omitting azimuth.

### Example: Projected point at 100,000 meters and bearing 45 degrees

```
SELECT ST_AsText(ST_Project('POINT(0 0)::geography', 100000, radians(45.0)));

POINT(0.635231029125537 0.639472334729198)
```

## Ver también

[ST\\_Azimuth](#), [ST\\_Distance](#), [PostgreSQL function radians\(\)](#)

## 7.5.23 ST\_QuantizeCoordinates

**ST\_QuantizeCoordinates** — Sets least significant bits of coordinates to zero

### Synopsis

geometry **ST\_QuantizeCoordinates** ( geometry g , int prec\_x , int prec\_y , int prec\_z , int prec\_m );

### Descripción

**ST\_QuantizeCoordinates** determines the number of bits (N) required to represent a coordinate value with a specified number of digits after the decimal point, and then sets all but the N most significant bits to zero. The resulting coordinate value will still round to the original value, but will have improved compressibility. This can result in a significant disk usage reduction provided that the geometry column is using a [compressible storage type](#). The function allows specification of a different number of digits after the decimal point in each dimension; unspecified dimensions are assumed to have the precision of the x dimension. Negative digits are interpreted to refer digits to the left of the decimal point, (i.e., `prec_x=-2` will preserve coordinate values to the nearest 100).

The coordinates produced by **ST\_QuantizeCoordinates** are independent of the geometry that contains those coordinates and the relative position of those coordinates within the geometry. As a result, existing topological relationships between geometries are unaffected by use of this function. The function may produce invalid geometry when it is called with a number of digits lower than the intrinsic precision of the geometry.

Availability: 2.5.0



Technical Background

PostGIS stores all coordinate values as double-precision floating point integers, which can reliably represent 15 significant digits. However, PostGIS may be used to manage data that intrinsically has fewer than 15 significant digits. An example is TIGER data, which is provided as geographic coordinates with six digits of precision after the decimal point (thus requiring only nine significant digits of longitude and eight significant digits of latitude.)

When 15 significant digits are available, there are many possible representations of a number with 9 significant digits. A double precision floating point number uses 52 explicit bits to represent the significand (mantissa) of the coordinate. Only 30 bits are needed to represent a mantissa with 9 significant digits, leaving 22 insignificant bits; we can set their value to anything we like and still end up with a number that rounds to our input value. For example, the value 100.123456 can be represented by the floating point numbers closest to 100.123456000000, 100.123456000001, and 100.123456432199. All are equally valid, in that `ST_AsText(geom, 6)` will return the same result with any of these inputs. As we can set these bits to any value, `ST_QuantizeCoordinates` sets the 22 insignificant bits to zero. For a long coordinate sequence this creates a pattern of blocks of consecutive zeros that is compressed by PostgreSQL more effeciently.



**Note**  
Only the on-disk size of the geometry is potentially affected by `ST_QuantizeCoordinates`. `ST_MemSize`, which reports the in-memory usage of the geometry, will return the the same value regardless of the disk space used by a geometry.

Ejemplos

```
SELECT ST_AsText(ST_QuantizeCoordinates('POINT (100.123456 0)::geometry, 4));
st_astext

POINT(100.123455047607 0)
```

```
WITH test AS (SELECT 'POINT (123.456789123456 123.456789123456)::geometry AS geom)
SELECT
 digits,
 encode(ST_QuantizeCoordinates(geom, digits), 'hex'),
 ST_AsText(ST_QuantizeCoordinates(geom, digits))
FROM test, generate_series(15, -15, -1) AS digits;
```

| digits | encode                                     | st_astext                                |
|--------|--------------------------------------------|------------------------------------------|
| 15     | 01010000005f9a72083cdd5e405f9a72083cdd5e40 | POINT(123.456789123456 123.456789123456) |
| 14     | 01010000005f9a72083cdd5e405f9a72083cdd5e40 | POINT(123.456789123456 123.456789123456) |
| 13     | 01010000005f9a72083cdd5e405f9a72083cdd5e40 | POINT(123.456789123456 123.456789123456) |
| 12     | 01010000005c9a72083cdd5e405c9a72083cdd5e40 | POINT(123.456789123456 123.456789123456) |
| 11     | 0101000000409a72083cdd5e40409a72083cdd5e40 | POINT(123.456789123456 123.456789123456) |
| 10     | 0101000000009a72083cdd5e40009a72083cdd5e40 | POINT(123.456789123455 123.456789123455) |
| 9      | 0101000000009072083cdd5e40009072083cdd5e40 | POINT(123.456789123418 123.456789123418) |
| 8      | 0101000000008072083cdd5e40008072083cdd5e40 | POINT(123.45678912336 123.45678912336)   |
| 7      | 0101000000000070083cdd5e40000070083cdd5e40 | POINT(123.456789121032 123.456789121032) |
| 6      | 0101000000000040083cdd5e40000040083cdd5e40 | POINT(123.456789076328 123.456789076328) |

|     |                                                                                           |
|-----|-------------------------------------------------------------------------------------------|
| 5   | 01010000000000000083cdd5e400000000083cdd5e40   POINT(123.456789016724 123.456789016724) ← |
| 4   | 01010000000000000003cdd5e400000000003cdd5e40   POINT(123.456787109375 123.456787109375) ← |
| 3   | 01010000000000000003cdd5e400000000003cdd5e40   POINT(123.456787109375 123.456787109375) ← |
| 2   | 010100000000000000038dd5e4000000000038dd5e40   POINT(123.45654296875 123.45654296875) ←   |
| 1   | 01010000000000000000dd5e400000000000dd5e40   POINT(123.453125 123.453125)                 |
| 0   | 01010000000000000000dc5e400000000000dc5e40   POINT(123.4375 123.4375)                     |
| -1  | 01010000000000000000c05e400000000000c05e40   POINT(123 123)                               |
| -2  | 01010000000000000000005e400000000000005e40   POINT(120 120)                               |
| -3  | 010100000000000000000058400000000000005840   POINT(96 96)                                 |
| -4  | 010100000000000000000058400000000000005840   POINT(96 96)                                 |
| -5  | 010100000000000000000058400000000000005840   POINT(96 96)                                 |
| -6  | 010100000000000000000058400000000000005840   POINT(96 96)                                 |
| -7  | 010100000000000000000058400000000000005840   POINT(96 96)                                 |
| -8  | 010100000000000000000058400000000000005840   POINT(96 96)                                 |
| -9  | 010100000000000000000058400000000000005840   POINT(96 96)                                 |
| -10 | 010100000000000000000058400000000000005840   POINT(96 96)                                 |
| -11 | 010100000000000000000058400000000000005840   POINT(96 96)                                 |
| -12 | 010100000000000000000058400000000000005840   POINT(96 96)                                 |
| -13 | 010100000000000000000058400000000000005840   POINT(96 96)                                 |
| -14 | 010100000000000000000058400000000000005840   POINT(96 96)                                 |
| -15 | 010100000000000000000058400000000000005840   POINT(96 96)                                 |

### Ver también

ST\_SnapToGrid

### 7.5.24 ST\_RemovePoint

**ST\_RemovePoint** — Remove a point from a linestring.

## Synopsis

```
geometry ST_RemovePoint(geometry linestring, integer offset);
```

### Descripción

Removes a point from a LineString, given its index (0-based). Useful for turning a closed line (ring) into an open linestring.

Enhanced: 3.2.0

Disponibilidad: 1.1.0



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

## Ejemplos

Guarantees no lines are closed by removing the end point of closed lines (rings). Assumes geom is of type LINESTRING

```
UPDATE sometable
 SET geom = ST_RemovePoint(geom, ST_NPoints(geom) - 1)
FROM sometable
WHERE ST_IsClosed(geom);
```

**Ver también**

[ST\\_AddPoint](#), [ST\\_NPoints](#), [ST\\_NumPoints](#)

**7.5.25 ST\_RemoveRepeatedPoints**

**ST\_RemoveRepeatedPoints** — Returns a version of a geometry with duplicate points removed.

**Synopsis**

geometry **ST\_RemoveRepeatedPoints**(geometry geom, float8 tolerance);

**Descripción**

Returns a version of the given geometry with duplicate consecutive points removed. The function processes only (Multi)LineStrings, (Multi)Polygons and MultiPoints but it can be called with any kind of geometry. Elements of GeometryCollections are processed individually. The endpoints of LineStrings are preserved.

If the *tolerance* parameter is provided, vertices within the tolerance distance of one another are considered to be duplicates.

Enhanced: 3.2.0

Disponibilidad: 2.2.0



This function supports Polyhedral surfaces.



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

**Ejemplos**

```
SELECT ST_AsText(ST_RemoveRepeatedPoints('MULTIPOINT ((1 1), (2 2), (3 3), (2 2))'));

MULTIPOINT(1 1,2 2,3 3)
```

```
SELECT ST_AsText(ST_RemoveRepeatedPoints('LINESTRING (0 0, 0 0, 1 1, 0 0, 1 1, 2 2)'));

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

**Example:** Collection elements are processed individually.

```
SELECT ST_AsText(ST_RemoveRepeatedPoints('GEOMETRYCOLLECTION (LINESTRING (1 1, 2 2, 2 2, ↵
3 3), POINT (4 4), POINT (4 4), POINT (5 5))'));

GEOMETRYCOLLECTION(LINESTRING(1 1,2 2,3 3),POINT(4 4),POINT(4 4),POINT(5 5))
```

**Example:** Repeated point removal with a distance tolerance.

```
SELECT ST_AsText(ST_RemoveRepeatedPoints('LINESTRING (0 0, 0 0, 1 1, 5 5, 1 1, 2 2)', 2)) ↵
;

LINESTRING(0 0,5 5,2 2)
```

**Ver también**

[ST\\_Simplify](#)

## 7.5.26 ST\_Reverse

**ST\_Reverse** — Devuelve la geometría con el orden de vértice invertido.

### Synopsis

geometry **ST\_Reverse**(geometry g1);

### Descripción

Se puede utilizar en cualquier geometría e invierte el orden de los vértices.

Mejorada: 2.4.0 se introdujo el soporte para curvas.



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



This function supports Polyhedral surfaces.

### Ejemplos

```
SELECT ST_AsText (geom) as line, ST_AsText (ST_Reverse (geom)) As reverseline
FROM
 (SELECT ST_MakeLine (ST_Point (1,2),
 ST_Point (1,10)) As geom) as foo;
--result
 line | reverseline
-----+-----
LINESTRING(1 2,1 10) | LINESTRING(1 10,1 2)
```

## 7.5.27 ST\_Segmentize

**ST\_Segmentize** — Returns a modified geometry/geography having no segment longer than a given distance.

### Synopsis

geometry **ST\_Segmentize**(geometry geom, float max\_segment\_length);  
 geography **ST\_Segmentize**(geography geog, float max\_segment\_length);

### Descripción

Returns a modified geometry/geography having no segment longer than `max_segment_length`. Length is computed in 2D. Segments are always split into equal-length subsegments.

- For geometry, the maximum length is in the units of the spatial reference system.
- For geography, the maximum length is in meters. Distances are computed on the sphere. Added vertices are created along the spherical great-circle arcs defined by segment endpoints.



#### Note

This only shortens long segments. It does not lengthen segments shorter than the maximum length.

**Warning**

For inputs containing long segments, specifying a relatively short `max_segment_length` can cause a very large number of vertices to be added. This can happen unintentionally if the argument is specified accidentally as a number of segments, rather than a maximum length.

Disponibilidad: 1.2.2

Enhanced: 3.0.0 Segmentize geometry now produces equal-length subsegments

Enhanced: 2.3.0 Segmentize geography now produces equal-length subsegments

Mejorada: 2.1.0 se introdujo el soporte para geography.

Changed: 2.1.0 As a result of the introduction of geography support, the usage `ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5)` causes an ambiguous function error. The input needs to be properly typed as a geometry or geography. Use `ST_GeomFromText`, `ST_GeogFromText` or a cast to the required type (e.g. `ST_Segmentize('LINESTRING(1 2, 3 4)::geometry, 0.5)` )

**Ejemplos**

Segmentizing a line. Long segments are split evenly, and short segments are not split.

```
SELECT ST_AsText(ST_Segmentize(
 'MULTILINESTRING((0 0, 0 1, 0 9), (1 10, 1 18))'::geometry,
 5));

MULTILINESTRING((0 0,0 1,0 5,0 9), (1 10,1 14,1 18))
```

Segmentizing a polygon:

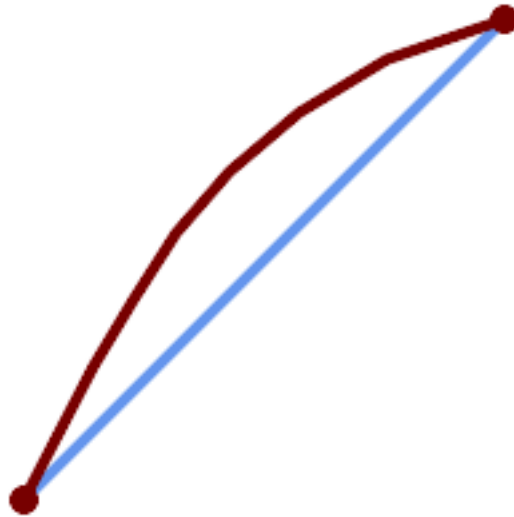
```
SELECT ST_AsText(
 ST_Segmentize(('POLYGON((0 0, 0 8, 30 0, 0 0))'::geometry), 10));

POLYGON((0 0,0 8,7.5 6,15 4,22.5 2,30 0,20 0,10 0,0 0))
```

Segmentizing a geographic line, using a maximum segment length of 2000 kilometers. Vertices are added along the great-circle arc connecting the endpoints.

```
SELECT ST_AsText(
 ST_Segmentize(('LINESTRING (0 0, 60 60)'::geography), 2000000));

LINESTRING(0 0,4.252632294621186 8.43596525986862,8.69579947419404 ↵
 16.824093489701564,13.550465473227048 25.107950473646188,19.1066053508691 ↵
 33.21091076089908,25.779290201459894 41.01711439406505,34.188839517966954 ↵
 48.337222885886,45.238153936612264 54.84733442373889,60 60)
```



*A geographic line segmentized along a great circle arc*

**Ver también**

[ST\\_LineSubstring](#)

## 7.5.28 ST\_SetPoint

**ST\_SetPoint** — Reemplace el punto de una cadena de línea con un punto dado.

### Synopsis

geometry **ST\_SetPoint**(geometry linestring, integer zerobasedposition, geometry point);

### Descripción

Reemplace el punto N de una cadena de línea con el punto dado. El índice comienza en 0. El índice negativo se cuenta hacia atrás, por lo que -1 es el último punto. Esto es especialmente útil en los disparadores cuando se trata de mantener la relación de las articulaciones cuando un vértice se mueve.

Disponibilidad: 1.1.0

Actualizado 2.3.0: indexación negativa



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

### Ejemplos

```
--Change first point in line string from -1 3 to -1 1
SELECT ST_AsText(ST_SetPoint('LINESTRING(-1 2,-1 3)', 0, 'POINT(-1 1)'));
 st_astext

LINESTRING(-1 1,-1 3)

---Change last point in a line string (lets play with 3d linestring this time)
SELECT ST_AsEWKT(ST_SetPoint(foo.geom, ST_NumPoints(foo.geom) - 1, ST_GeomFromEWKT('POINT ↵
(-1 1 3)')))
```

```

FROM (SELECT ST_GeomFromEWKT('LINESTRING(-1 2 3,-1 3 4, 5 6 7)') As geom) As foo;
 st_asewkt

LINESTRING(-1 2 3,-1 3 4,-1 1 3)

SELECT ST_AsText(ST_SetPoint(g, -3, p))
FROM ST_GeomFromText('LINESTRING(0 0, 1 1, 2 2, 3 3, 4 4)') AS g
 , ST_PointN(g,1) as p;
 st_astext

LINESTRING(0 0,1 1,0 0,3 3,4 4)

```

### Ver también

[ST\\_AddPoint](#), [ST\\_NPoints](#), [ST\\_NumPoints](#), [ST\\_PointN](#), [ST\\_RemovePoint](#)

## 7.5.29 ST\_ShiftLongitude

**ST\_ShiftLongitude** — Shifts the longitude coordinates of a geometry between -180..180 and 0..360.

### Synopsis

geometry **ST\_ShiftLongitude**(geometry geom);

### Descripción

Reads every point/vertex in a geometry, and shifts its longitude coordinate from -180..0 to 180..360 and vice versa if between these ranges. This function is symmetrical so the result is a 0..360 representation of a -180..180 data and a -180..180 representation of a 0..360 data.



#### Note

This is only useful for data with coordinates in longitude/latitude; e.g. SRID 4326 (WGS 84 geographic)



#### Warning

Pre-1.3.4 bug prevented this from working for MULTIPOINT. 1.3.4+ works with MULTIPOINT as well.



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

Mejora: 2.0.0 se introdujeron soporte de superficies poliédricas y TIN.

NOTE: this function was renamed from "ST\_Shift\_Longitude" in 2.2.0



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Ejemplos

```
--single point forward transformation
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(270 0)::geometry'))

st_astext

POINT(-90 0)

--single point reverse transformation
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(-90 0)::geometry'))

st_astext

POINT(270 0)

--for linestrings the functions affects only to the sufficient coordinates
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;LINESTRING(174 12, 182 13)::geometry'))

st_astext

LINESTRING(174 12,-178 13)
```

## Ver también

[ST\\_WrapX](#)

### 7.5.30 ST\_WrapX

ST\_WrapX — Wrap a geometry around an X value.

#### Synopsis

geometry **ST\_WrapX**(geometry geom, float8 wrap, float8 move);

#### Descripción

This function splits the input geometries and then moves every resulting component falling on the right (for negative 'move') or on the left (for positive 'move') of given 'wrap' line in the direction specified by the 'move' parameter, finally re-unioning the pieces together.



#### Note

This is useful to "recenter" long-lat input to have features of interest not spawned from one side to the other.

Availability: 2.3.0 requires GEOS



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



## Ejemplos

```
-- Move all components of the given geometries whose bounding box
-- falls completely on the left of x=0 to +360
select ST_WrapX(geom, 0, 360);

-- Move all components of the given geometries whose bounding box
-- falls completely on the left of x=-30 to +360
select ST_WrapX(geom, -30, 360);
```

## Ver también

[ST\\_ShiftLongitude](#)

### 7.5.31 ST\_SnapToGrid

ST\_SnapToGrid — Ajusta todos los puntos de la geometría de entrada a una cuadrícula regular.

## Synopsis

```
geometry ST_SnapToGrid(geometry geomA, float originX, float originY, float sizeX, float sizeY);
geometry ST_SnapToGrid(geometry geomA, float sizeX, float sizeY);
geometry ST_SnapToGrid(geometry geomA, float size);
geometry ST_SnapToGrid(geometry geomA, geometry pointOrigin, float sizeX, float sizeY, float sizeZ, float sizeM);
```

## Descripción

Variante 1, 2, 3: ajusta todos los puntos de la geometría de entrada a la cuadrícula definida por su origen y tamaño de celda. Elimina los puntos consecutivos que caen en la misma celda, eventualmente devuelve NULL si los puntos de salida no son suficientes para definir una geometría del tipo dado. Las geometrías contraídas de una colección se despojan de ella. Útil para reducir la precisión.

Variante 4: introducido 1.1.0 - Ajusta todos los puntos de la geometría de entrada a la cuadrícula definida por su origen (el segundo argumento, debe ser un punto) y tamaños de celda. Especifique 0 como tamaño para cualquier dimensión que no desee ajustar a una cuadrícula.



#### Note

La geometría devuelta podría perder su simplicidad (ver [ST\\_IsSimple](#)).



#### Note

Antes del lanzamiento 1.1.0 esta función siempre devolvió una geometría 2d. A partir de 1.1.0 la geometría devuelta tendrá la misma dimensionalidad que la entrada con valores de dimensión más altos sin tocar. Utilice la versión que toma un segundo argumento de geometría para definir todas las dimensiones de cuadrícula.

Disponibilidad: 1.0.0RC1

Disponibilidad: 1.1.0 - soporte de Z y M



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

## Ejemplos

```
--Snap your geometries to a precision grid of 10^-3
UPDATE mytable
 SET geom = ST_SnapToGrid(geom, 0.001);

SELECT ST_AsText(ST_SnapToGrid(
 ST_GeomFromText('LINESTRING(1.1115678 2.123, 4.111111 3.2374897, ↵
 4.11112 3.23748667)'),
 0.001)
);
 st_astext

LINESTRING(1.112 2.123,4.111 3.237)
--Snap a 4d geometry
SELECT ST_AsEWKT(ST_SnapToGrid(
 ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 2.3456 1.11111,
 4.111111 3.2374897 3.1234 1.1111, -1.11111112 2.123 2.3456 1.111112)'),
 ST_GeomFromEWKT('POINT(1.12 2.22 3.2 4.4444)'),
 0.1, 0.1, 0.1, 0.01));
 st_asewkt

LINESTRING(-1.08 2.12 2.3 1.1144,4.12 3.22 3.1 1.1144,-1.08 2.12 2.3 1.1144)

--With a 4d geometry - the ST_SnapToGrid(geom,size) only touches x and y coords but keeps m ↵
and z the same
SELECT ST_AsEWKT(ST_SnapToGrid(ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 3 2.3456,
 4.111111 3.2374897 3.1234 1.1111)'),
 0.01)
);
 st_asewkt

LINESTRING(-1.11 2.12 3 2.3456,4.11 3.24 3.1234 1.1111)
```

## Ver también

[ST\\_Snap](#), [ST\\_AsEWKT](#), [ST\\_AsText](#), [ST\\_GeomFromText](#), [ST\\_GeomFromEWKT](#), [ST\\_Simplify](#)

## 7.5.32 ST\_Snap

**ST\_Snap** — Ajusta segmentos y vértices de la geometría de entrada a vértices de una geometría de referencia.

### Synopsis

geometry **ST\_Snap**(geometry input, geometry reference, float tolerance);

### Descripción

Snaps the vertices and segments of a geometry to another Geometry's vertices. A snap distance tolerance is used to control where snapping is performed. The result geometry is the input geometry with the vertices snapped. If no snapping occurs then the input geometry is returned unchanged.

El ajustar una geometría a otra puede mejorar la robustez de las operaciones de superposición eliminando los bordes casi coincidentes (que causan problemas durante el cálculo de noding y de intersección).

Un ajuste excesivo puede resultar en la creación de una topología no válida, por lo que el número y la ubicación de los vértices ajustados se deciden usando heurísticas para determinar cuándo es seguro ajustar. Sin embargo, esto puede resultar en que algunos potenciales ajustes se omitan.

**Note**

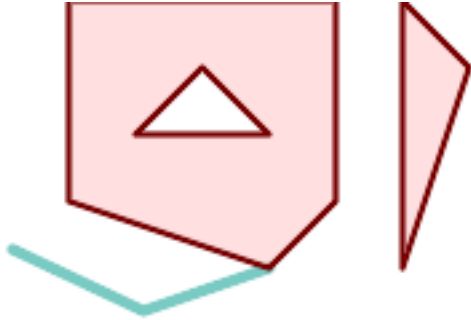
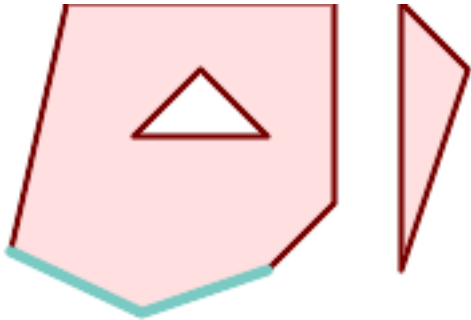
La geometría devuelta puede perder su simplicidad (ver [ST\\_IsSimple](#)) y su validez (ver [ST\\_IsValid](#)).

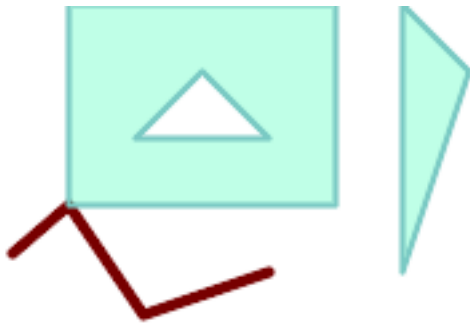
Realizado por el módulo GEOS.

Disponibilidad: 2.0.0

**Ejemplos**

*Un multipolígono mostrado con una cadena de líneas (antes de cualquier ajuste)*

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <p><i>Un multipolígono se ajustó a una cadena de línea a la tolerancia: 1,01 de distancia. El nuevo multipolígono se muestra en referencia a la cadena de línea</i></p> <pre>SELECT ST_AsText(ST_Snap(poly,line, ↵     ST_Distance(poly,line)*1.01)) AS polysnapped FROM (SELECT     ST_GeomFromText('MULTIPOLYGON(         ((26 125, 26 200, 126 200, 126 125, ↵         26 125 ),         ( 51 150, 101 150, 76 175, 51 150 ) ↵     ),     (( 151 100, 151 200, 176 175, 151 ↵     100 )))') As poly,     ST_GeomFromText('LINESTRING (5 ↵     107, 54 84, 101 100)') As line     ) As foo;</pre> <div>polysnapped</div> | <p><i>Un multipolígono se ajustó a una cadena de línea a la tolerancia: 1,25 de distancia. El nuevo multipolígono se muestra en referencia a la cadena de línea</i></p> <pre>SELECT ST_AsText(     ST_Snap(poly,line, ST_Distance(poly, ↵     line)*1.25)     ) AS polysnapped FROM (SELECT     ST_GeomFromText('MULTIPOLYGON(         (( 26 125, 26 200, 126 200, 126 125, ↵         26 125 ),         ( 51 150, 101 150, 76 175, 51 150 ) ↵     ),     (( 151 100, 151 200, 176 175, 151 ↵     100 )))') As poly,     ST_GeomFromText('LINESTRING (5 ↵     107, 54 84, 101 100)') As line     ) As foo;</pre> <div>polysnapped</div> |
| <pre>MULTIPOLYGON(((26 125,26 200,126 200,126 ↵     125,101 100,26 125),     (51 150,101 150,76 175,51 150)),((151 ↵     100,151 200,176 175,151 100)))</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                              | <pre>MULTIPOLYGON(((5 107,26 200,126 200,126 ↵     125,101 100,54 84,5 107),     (51 150,101 150,76 175,51 150)),((151 ↵     100,151 200,176 175,151 100)))</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

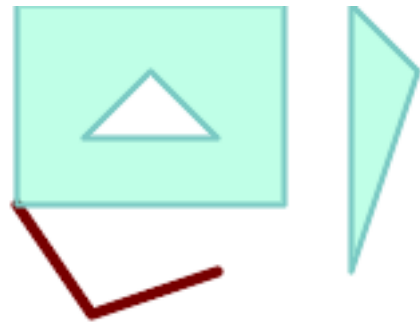


*La cadena de línea se ajustó al multipolígono original a la tolerancia 1,01 de distancia. La nueva cadena de línea se muestra con referencia al multipolígono*

```
SELECT ST_AsText (
 ST_Snap(line, poly, ST_Distance(poly, ↵
 line)*1.01)
) AS linesnapped
FROM (SELECT
 ST_GeomFromText('MULTIPOLYGON(
 ((26 125, 26 200, 126 200, 126 125, ↵
 26 125),
 (51 150, 101 150, 76 175, 51 150)) ↵
 ',
 ((151 100, 151 200, 176 175, 151 ↵
 100)))') As poly,
 ST_GeomFromText('LINESTRING (5 ↵
 107, 54 84, 101 100)') As line
) As foo;

 linesnapped

LINESTRING(5 107,26 125,54 84,101 100)
```



*La cadena de línea se ajustó al multipolígono original a la tolerancia 1,25 de distancia. La nueva cadena de línea se muestra con referencia al multipolígono*

```
SELECT ST_AsText (
 ST_Snap(line, poly, ST_Distance(poly, ↵
 line)*1.25)
) AS linesnapped
FROM (SELECT
 ST_GeomFromText('MULTIPOLYGON(
 ((26 125, 26 200, 126 200, 126 125, ↵
 26 125),
 (51 150, 101 150, 76 175, 51 150)) ↵
 ',
 ((151 100, 151 200, 176 175, 151 ↵
 100)))') As poly,
 ST_GeomFromText('LINESTRING (5 ↵
 107, 54 84, 101 100)') As line
) As foo;

 linesnapped

LINESTRING(26 125,54 84,101 100)
```

**Ver también**

[ST\\_SnapToGrid](#)

### 7.5.33 ST\_SwapOrdinates

**ST\_SwapOrdinates** — Returns a version of the given geometry with given ordinate values swapped.

#### Synopsis

geometry **ST\_SwapOrdinates**(geometry geom, cstring ords);

## Descripción

Returns a version of the given geometry with given ordinates swapped.

The `ords` parameter is a 2-characters string naming the ordinates to swap. Valid names are: x,y,z and m.

Disponibilidad: 2.2.0



This method supports Circular Strings and Curves.



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



This function supports M coordinates.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Ejemplo

```
-- Scale M value by 2
SELECT ST_AsText(
 ST_SwapOrdinates(
 ST_Scale(
 ST_SwapOrdinates(g, 'xm'),
 2, 1
),
 'xm'
)
) FROM (SELECT 'POINT ZM (0 0 0 2)::geometry g) foo;
 st_astext

POINT ZM (0 0 0 4)
```

## Ver también

[ST\\_FlipCoordinates](#)

## 7.6 Geometry Validation

### 7.6.1 ST\_IsValid

`ST_IsValid` — Tests if a geometry is well-formed in 2D.

#### Synopsis

```
boolean ST_IsValid(geometry g);
boolean ST_IsValid(geometry g, integer flags);
```

## Description

Tests if an `ST_Geometry` value is well-formed and valid in 2D according to the OGC rules. For geometries with 3 and 4 dimensions, the validity is still only tested in 2 dimensions. For geometries that are invalid, a PostgreSQL NOTICE is emitted providing details of why it is not valid.

For the version with the `flags` parameter, supported values are documented in [ST\\_IsValidDetail](#). This version does not print a NOTICE explaining invalidity.

For more information on the definition of geometry validity, refer to [Section 4.4](#).



### Note

SQL-MM defines the result of `ST_IsValid(NULL)` to be 0, while PostGIS returns NULL.

Performed by the GEOS module.

The version accepting flags is available starting with 2.0.0.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 5.1.9



### Note

Neither OGC-SFS nor SQL-MM specifications include a flag argument for `ST_IsValid`. The flag is a PostGIS extension.

## Examples

```
SELECT ST_IsValid(ST_GeomFromText('LINESTRING(0 0, 1 1)')) As good_line,
 ST_IsValid(ST_GeomFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) As bad_poly
--results
NOTICE: Self-intersection at or near point 0 0
good_line | bad_poly
-----+-----
t | f
```

## See Also

[ST\\_IsSimple](#), [ST\\_IsValidReason](#), [ST\\_IsValidDetail](#),

### 7.6.2 ST\_IsValidDetail

`ST_IsValidDetail` — Returns a `valid_detail` row stating if a geometry is valid or if not a reason and a location.

## Synopsis

`valid_detail` **ST\_IsValidDetail**(geometry geom, integer flags);

## Description

Returns a `valid_detail` row, containing a boolean (`valid`) stating if a geometry is valid, a varchar (`reason`) stating a reason why it is invalid and a geometry (`location`) pointing out where it is invalid.

Useful to improve on the combination of `ST_IsValid` and `ST_IsValidReason` to generate a detailed report of invalid geometries.

The optional `flags` parameter is a bitfield. It can have the following values:

- 0: Use usual OGC SFS validity semantics.
- 1: Consider certain kinds of self-touching rings (inverted shells and exverted holes) as valid. This is also known as "the ESRI flag", since this is the validity model used by those tools. Note that this is invalid under the OGC model.

Performed by the GEOS module.

Availability: 2.0.0

## Examples

```
--First 3 Rejects from a successful quintuplet experiment
SELECT gid, reason(ST_IsValidDetail(geom)), ST_AsText(location(ST_IsValidDetail(geom))) as ←
 location
FROM
 (SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As geom, gid
 FROM (SELECT ST_Buffer(ST_Point(x1*10,y1), z1) As buff, x1*10 + y1*100 + z1*1000 As gid
 FROM generate_series(-4,6) x1
 CROSS JOIN generate_series(2,5) y1
 CROSS JOIN generate_series(1,8) z1
 WHERE x1 > y1*0.5 AND z1 < x1*y1) As e
 INNER JOIN (SELECT ST_Translate(ST_ExteriorRing(ST_Buffer(ST_Point(x1*10,y1), z1)), ←
 y1*1, z1*2) As line
 FROM generate_series(-3,6) x1
 CROSS JOIN generate_series(2,5) y1
 CROSS JOIN generate_series(1,10) z1
 WHERE x1 > y1*0.75 AND z1 < x1*y1) As f
 ON (ST_Area(e.buff) > 78 AND ST_Contains(e.buff, f.line))
 GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(geom) = false
ORDER BY gid
LIMIT 3;
```

| gid  | reason            | location    |
|------|-------------------|-------------|
| 5330 | Self-intersection | POINT(32 5) |
| 5340 | Self-intersection | POINT(42 5) |
| 5350 | Self-intersection | POINT(52 5) |

```
--simple example
SELECT * FROM ST_IsValidDetail('LINESTRING(220227 150406,220227 150407,22020 150410)');
```

| valid | reason | location |
|-------|--------|----------|
| t     |        |          |

## See Also

[ST\\_IsValid](#), [ST\\_IsValidReason](#)



### 7.6.3 ST\_IsValidReason

**ST\_IsValidReason** — Returns text stating if a geometry is valid, or a reason for invalidity.

#### Synopsis

```
text ST_IsValidReason(geometry geomA);
text ST_IsValidReason(geometry geomA, integer flags);
```

#### Description

Returns text stating if a geometry is valid, or if invalid a reason why.

Useful in combination with **ST\_IsValid** to generate a detailed report of invalid geometries and reasons.

Allowed flags are documented in **ST\_IsValidDetail**.

Performed by the GEOS module.

Availability: 1.4

Availability: 2.0 version taking flags.

#### Examples

```
-- invalid bow-tie polygon
SELECT ST_IsValidReason(
 'POLYGON ((100 200, 100 100, 200 200,
 200 100, 100 200))'::geometry) as validity_info;
validity_info

Self-intersection[150 150]
```

```
--First 3 Rejects from a successful quintuplet experiment
SELECT gid, ST_IsValidReason(geom) as validity_info
FROM
 (SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As geom, gid
 FROM (SELECT ST_Buffer(ST_Point(x1*10,y1), z1) As buff, x1*10 + y1*100 + z1*1000 As gid
 FROM generate_series(-4,6) x1
 CROSS JOIN generate_series(2,5) y1
 CROSS JOIN generate_series(1,8) z1
 WHERE x1 > y1*0.5 AND z1 < x1*y1) As e
 INNER JOIN (SELECT ST_Translate(ST_ExteriorRing(ST_Buffer(ST_Point(x1*10,y1), z1)),
 y1*1, z1*2) As line
 FROM generate_series(-3,6) x1
 CROSS JOIN generate_series(2,5) y1
 CROSS JOIN generate_series(1,10) z1
 WHERE x1 > y1*0.75 AND z1 < x1*y1) As f
 ON (ST_Area(e.buff) > 78 AND ST_Contains(e.buff, f.line))
 GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(geom) = false
ORDER BY gid
LIMIT 3;

gid | validity_info
-----+-----
5330 | Self-intersection [32 5]
5340 | Self-intersection [42 5]
5350 | Self-intersection [52 5]
```

```
--simple example
SELECT ST_IsValidReason('LINESTRING(220227 150406,220227 150407,222020 150410)');

st_isvalidreason

Valid Geometry
```

### See Also

[ST\\_IsValid](#), [ST\\_Summary](#)

## 7.6.4 ST\_MakeValid

**ST\_MakeValid** — Attempts to make an invalid geometry valid without losing vertices.

### Synopsis

```
geometry ST_MakeValid(geometry input);
geometry ST_MakeValid(geometry input, text params);
```

### Description

The function attempts to create a valid representation of a given invalid geometry without losing any of the input vertices. Valid geometries are returned unchanged.

Supported inputs are: POINTS, MULTIPOINTS, LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS and GEOMETRYCOLLECTIONS containing any mix of them.

In case of full or partial dimensional collapses, the output geometry may be a collection of lower-to-equal dimension geometries, or a geometry of lower dimension.

Single polygons may become multi-geometries in case of self-intersections.

The `params` argument can be used to supply an options string to select the method to use for building valid geometry. The options string is in the format "method=linework|structure keepcollapsed=truelfalse". If no "params" argument is provided, the "linework" algorithm will be used as the default.

The "method" key has two values.

- "linework" is the original algorithm, and builds valid geometries by first extracting all lines, noding that linework together, then building a value output from the linework.
- "structure" is an algorithm that distinguishes between interior and exterior rings, building new geometry by unioning exterior rings, and then differencing all interior rings.

The "keepcollapsed" key is only valid for the "structure" algorithm, and takes a value of "true" or "false". When set to "false", geometry components that collapse to a lower dimensionality, for example a one-point linestring would be dropped.

Performed by the GEOS module.

Availability: 2.0.0

Enhanced: 2.0.1, speed improvements

Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT.

Enhanced: 3.1.0, added removal of Coordinates with NaN values.

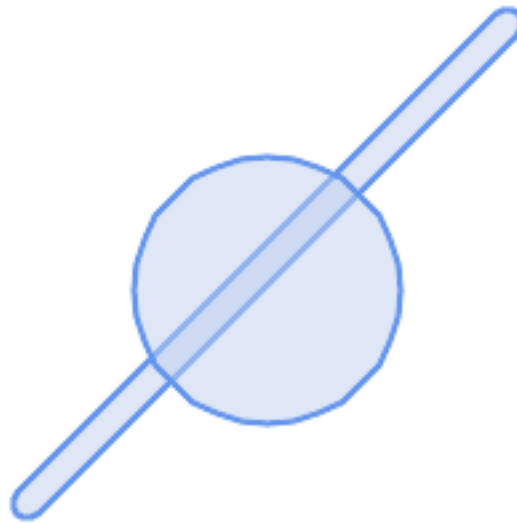
Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS >= 3.10.0.



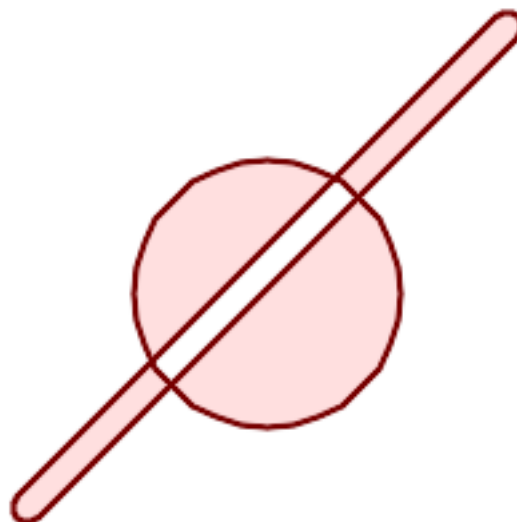
This function supports 3d and will not drop the z-index.

**Examples**

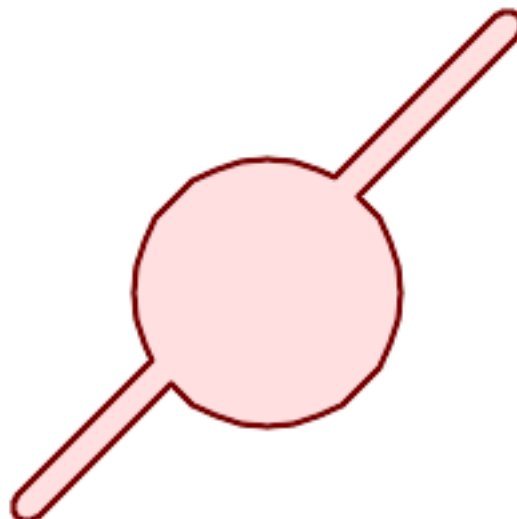
---



*before\_geom: MULTIPOLYGON of 2 overlapping polygons*



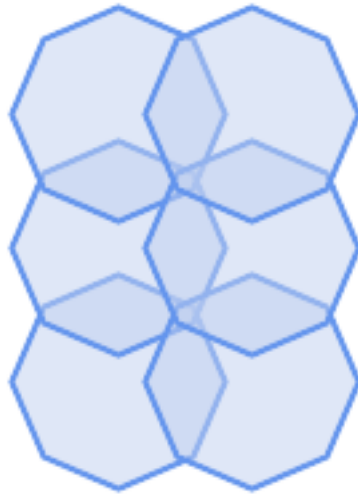
*after\_geom: MULTIPOLYGON of 4 non-overlapping polygons*



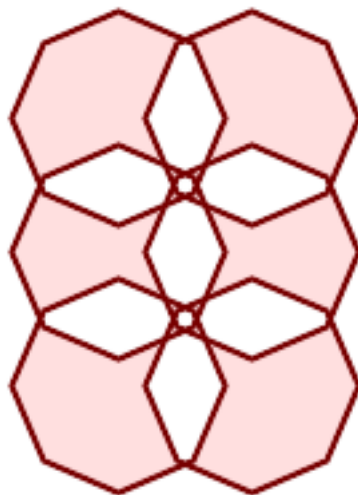
*after\_geom\_structure: MULTIPOLYGON of 1 non-overlapping polygon*

```
SELECT f.geom AS before_geom, ST_MakeValid(f.geom) AS after_geom, ST_MakeValid(f.geom, ←
 'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,
191 195 192 195 193 194 194 194 194 193 195 192 195 191
```

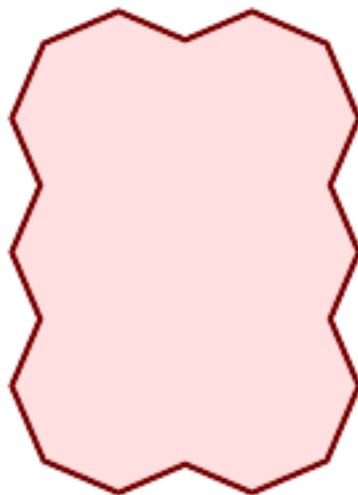




*before\_geom: MULTIPOLYGON of 6 overlapping polygons*



*after\_geom: MULTIPOLYGON of 14 Non-overlapping polygons*



*after\_geom\_structure: MULTIPOLYGON of 1 Non-overlapping polygon*

```
SELECT c.geom AS before_geom,
 ST_MakeValid(c.geom) AS after_geom,
 ST_MakeValid(c.geom, 'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((91 50,79 22,51 10,23 22,11 50,23 78,51 90,79 78,91 ↵
```

## Examples

```
SELECT ST_AsText(ST_MakeValid(
 'LINESTRING(0 0, 0 0)',
 'method=structure keepcollapsed=true'
));

st_astext

POINT(0 0)

SELECT ST_AsText(ST_MakeValid(
 'LINESTRING(0 0, 0 0)',
 'method=structure keepcollapsed=false'
));

st_astext

LINESTRING EMPTY
```

## See Also

[ST\\_IsValid](#), [ST\\_GeomCollFromText](#), [ST\\_CollectionExtract](#)

## 7.7 Spatial Reference System Functions

### 7.7.1 ST\_InverseTransformPipeline

**ST\_InverseTransformPipeline** — Return a new geometry with coordinates transformed to a different spatial reference system using the inverse of a defined coordinate transformation pipeline.

#### Synopsis

geometry **ST\_InverseTransformPipeline**(geometry geom, text pipeline, integer to\_srid);

#### Description

Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline to go in the inverse direction.

Refer to [ST\\_TransformPipeline](#) for details on writing a transformation pipeline.

Availability: 3.4.0

The SRID of the input geometry is ignored, and the SRID of the output geometry will be set to zero unless a value is provided via the optional `to_srid` parameter. When using [ST\\_TransformPipeline](#) the pipeline is executed in a forward direction. Using `ST_InverseTransformPipeline()` the pipeline is executed in the inverse direction.

Transforms using pipelines are a specialised version of [ST\\_Transform](#). In most cases `ST_Transform` will choose the correct operations to convert between coordinate systems, and should be preferred.

## Examples

Change WGS 84 long lat to UTM 31N using the EPSG:16031 conversion

```
-- Inverse direction
SELECT ST_AsText(ST_InverseTransformPipeline('POINT(426857.9877165967 5427937.523342293)'::geometry,
 'urn:ogc:def:coordinateOperation:EPSG::16031')) AS wgs_geom;

 wgs_geom

POINT(2 48.99999999999999)
(1 row)
```

GDA2020 example.

```
-- using ST_Transform with automatic selection of a conversion pipeline.
SELECT ST_AsText(ST_Transform('SRID=4939;POINT(143.0 -37.0)'::geometry, 7844)) AS gda2020_auto;

 gda2020_auto

POINT(143.00000635638918 -36.999986706128176)
(1 row)
```

## See Also

[ST\\_Transform](#), [ST\\_TransformPipeline](#)

## 7.7.2 ST\_SetSRID

ST\_SetSRID — Set the SRID on a geometry.

### Synopsis

geometry **ST\_SetSRID**(geometry geom, integer srid);

### Description

Sets the SRID on a geometry to a particular integer value. Useful in constructing bounding boxes for queries.



#### Note

This function does not transform the geometry coordinates in any way - it simply sets the meta data defining the spatial reference system the geometry is assumed to be in. Use [ST\\_Transform](#) if you want to transform the geometry into a new projection.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method supports Circular Strings and Curves.



## Examples

-- Mark a point as WGS 84 long lat --

```
SELECT ST_SetSRID(ST_Point(-123.365556, 48.428611),4326) As wgs84long_lat;
-- the ewkt representation (wrap with ST_AsEWKT) -
SRID=4326;POINT(-123.365556 48.428611)
```

-- Mark a point as WGS 84 long lat and then transform to web mercator (Spherical Mercator) --

```
SELECT ST_Transform(ST_SetSRID(ST_Point(-123.365556, 48.428611),4326),3785) As spere_merc;
-- the ewkt representation (wrap with ST_AsEWKT) -
SRID=3785;POINT(-13732990.8753491 6178458.96425423)
```

## See Also

Section [4.5](#), [ST\\_SRID](#), [ST\\_Transform](#), [UpdateGeometrySRID](#)

## 7.7.3 ST\_SRID

**ST\_SRID** — Returns the spatial reference identifier for a geometry.

### Synopsis

integer **ST\_SRID**(geometry g1);

### Description

Returns the spatial reference identifier for the ST\_Geometry as defined in spatial\_ref\_sys table. Section [4.5](#)



#### Note

spatial\_ref\_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



This method implements the SQL/MM specification. SQL-MM 3: 5.1.5



This method supports Circular Strings and Curves.

## Examples

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
--result
4326
```

## See Also

Section [4.5](#), [ST\\_SetSRID](#), [ST\\_Transform](#), [ST\\_SRID](#), [ST\\_SRID](#)

### 7.7.4 ST\_Transform

ST\_Transform — Return a new geometry with coordinates transformed to a different spatial reference system.

#### Synopsis

```
geometry ST_Transform(geometry g1, integer srid);
geometry ST_Transform(geometry geom, text to_proj);
geometry ST_Transform(geometry geom, text from_proj, text to_proj);
geometry ST_Transform(geometry geom, text from_proj, integer to_srid);
```

#### Description

Returns a new geometry with its coordinates transformed to a different spatial reference system. The destination spatial reference `to_srid` may be identified by a valid SRID integer parameter (i.e. it must exist in the `spatial_ref_sys` table). Alternatively, a spatial reference defined as a PROJ.4 string can be used for `to_proj` and/or `from_proj`, however these methods are not optimized. If the destination spatial reference system is expressed with a PROJ.4 string instead of an SRID, the SRID of the output geometry will be set to zero. With the exception of functions with `from_proj`, input geometries must have a defined SRID.

ST\_Transform is often confused with [ST\\_SetSRID](#). ST\_Transform actually changes the coordinates of a geometry from one spatial reference system to another, while ST\_SetSRID() simply changes the SRID identifier of the geometry.

ST\_Transform automatically selects a suitable conversion pipeline given the source and target spatial reference systems. To use a specific conversion method, use [ST\\_TransformPipeline](#).



#### Note

Requires PostGIS be compiled with PROJ support. Use [PostGIS\\_Full\\_Version](#) to confirm you have PROJ support compiled in.



#### Note

If using more than one transformation, it is useful to have a functional index on the commonly used transformations to take advantage of index usage.



#### Note

Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

Enhanced: 2.3.0 support for direct PROJ.4 text was introduced.



This method implements the SQL/MM specification. SQL-MM 3: 5.1.6



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

## Examples

### Change Massachusetts state plane US feet geometry to WGS 84 long lat

```
SELECT ST_AsText(ST_Transform(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
743265 2967450,743265.625 2967416,743238 2967416))',2249),4326)) As wgs_geom;

wgs_geom

POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.177684
8522251 42.3902896512902));
(1 row)

--3D Circular String example
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromEWKT('SRID=2249;CIRCULARSTRING(743238 2967416 ↵
1,743238 2967450 2,743265 2967450 3,743265.625 2967416 3,743238 2967416 4)'),4326));

st_asewkt

SRID=4326;CIRCULARSTRING(-71.1776848522251 42.3902896512902 1,-71.1776843766326 ↵
42.3903829478009 2,
-71.1775844305465 42.3903826677917 3,
-71.1775825927231 42.3902893647987 3,-71.1776848522251 42.3902896512902 4)
```

Example of creating a partial functional index. For tables where you are not sure all the geometries will be filled in, its best to use a partial index that leaves out null geometries which will both conserve space and make your index smaller and more efficient.

```
CREATE INDEX idx_geom_26986_parcel
ON parcels
USING gist
(ST_Transform(geom, 26986))
WHERE geom IS NOT NULL;
```

### Examples of using PROJ.4 text to transform with custom spatial references.

```
-- Find intersection of two polygons near the North pole, using a custom Gnomonic projection
-- See http://boundlessgeo.com/2012/02/flattening-the-peel/
WITH data AS (
SELECT
ST_GeomFromText('POLYGON((170 50,170 72,-130 72,-130 50,170 50))', 4326) AS p1,
ST_GeomFromText('POLYGON((-170 68,-170 90,-141 90,-141 68,-170 68))', 4326) AS p2,
'+proj=gnom +ellps=WGS84 +lat_0=70 +lon_0=-160 +no_defs'::text AS gnom
)
SELECT ST_AsText(
ST_Transform(
ST_Intersection(ST_Transform(p1, gnom), ST_Transform(p2, gnom)),
gnom, 4326))
FROM data;

st_astext

POLYGON((-170 74.053793645338,-141 73.4268621378904,-141 68,-170 68,-170 74.053793645338) ↵
)
```

## Configuring transformation behavior

Sometimes coordinate transformation involving a grid-shift can fail, for example if PROJ.4 has not been built with grid-shift files or the coordinate does not lie within the range for which the grid shift is defined. By default, PostGIS will throw an error if a

grid shift file is not present, but this behavior can be configured on a per-SRID basis either by testing different `to_proj` values of PROJ.4 text, or altering the `proj4text` value within the `spatial_ref_sys` table.

For example, the `proj4text` parameter `+datum=NAD87` is a shorthand form for the following `+nadgrids` parameter:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat
```

The `@` prefix means no error is reported if the files are not present, but if the end of the list is reached with no file having been appropriate (ie. found and overlapping) then an error is issued.

If, conversely, you wanted to ensure that at least the standard files were present, but that if all files were scanned without a hit a null transformation is applied you could use:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat,null
```

The null grid shift file is a valid grid shift file covering the whole world and applying no shift. So for a complete example, if you wanted to alter PostGIS so that transformations to SRID 4267 that didn't lie within the correct range did not throw an ERROR, you would use the following:

```
UPDATE spatial_ref_sys SET proj4text = '+proj=longlat +ellps=clrk66 +nadgrids=@conus, ↵
 @alaska,@ntv2_0.gsb,@ntv1_can.dat,null +no_defs' WHERE srid = 4267;
```

## See Also

Section 4.5, [ST\\_SetSRID](#), [ST\\_SRID](#), [UpdateGeometrySRID](#), [ST\\_TransformPipeline](#)

### 7.7.5 ST\_TransformPipeline

**ST\_TransformPipeline** — Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.

## Synopsis

```
geometry ST_TransformPipeline(geometry g1, text pipeline, integer to_srid);
```

## Description

Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.

Transformation pipelines are defined using any of the following string formats:

- `urn:ogc:def:coordinateOperation:AUTHORITY::CODE`. Note that a simple `EPSG:CODE` string does not uniquely identify a coordinate operation: the same EPSG code can be used for a CRS definition.
- A PROJ pipeline string of the form: `+proj=pipeline ...`. Automatic axis normalisation will not be applied, and if necessary the caller will need to add an additional pipeline step, or remove `axiswap` steps.
- Concatenated operations of the form: `urn:ogc:def:coordinateOperation,coordinateOperation:EPSG::3895,...`

Availability: 3.4.0

The SRID of the input geometry is ignored, and the SRID of the output geometry will be set to zero unless a value is provided via the optional `to_srid` parameter. When using `ST_TransformPipeline()` the pipeline is executed in a forward direction. Using [ST\\_InverseTransformPipeline](#) the pipeline is executed in the inverse direction.

Transforms using pipelines are a specialised version of [ST\\_Transform](#). In most cases `ST_Transform` will choose the correct operations to convert between coordinate systems, and should be preferred.

## Examples

### Change WGS 84 long lat to UTM 31N using the EPSG:16031 conversion

```
-- Forward direction
SELECT ST_AsText(ST_TransformPipeline('SRID=4326;POINT(2 49) '::geometry,
 'urn:ogc:def:coordinateOperation:EPSG::16031') AS utm_geom);

 utm_geom

POINT(426857.9877165967 5427937.523342293)
(1 row)

-- Inverse direction
SELECT ST_AsText(ST_InverseTransformPipeline('POINT(426857.9877165967 5427937.523342293) ':: ←
 geometry,
 'urn:ogc:def:coordinateOperation:EPSG::16031')) AS wgs_geom;

 wgs_geom

POINT(2 48.99999999999999)
(1 row)
```

### GDA2020 example.

```
-- using ST_Transform with automatic selection of a conversion pipeline.
SELECT ST_AsText(ST_Transform('SRID=4939;POINT(143.0 -37.0) '::geometry, 7844)) AS ←
 gda2020_auto;

 gda2020_auto

POINT(143.00000635638918 -36.999986706128176)
(1 row)

-- using a defined conversion (EPSG:8447)
SELECT ST_AsText(ST_TransformPipeline('SRID=4939;POINT(143.0 -37.0) '::geometry,
 'urn:ogc:def:coordinateOperation:EPSG::8447')) AS gda2020_code;

 gda2020_code

POINT(143.0000063280214 -36.999986718287545)
(1 row)

-- using a PROJ pipeline definition matching EPSG:8447, as returned from
-- 'projinfo -s EPSG:4939 -t EPSG:7844'.
-- NOTE: any 'axiswap' steps must be removed.
SELECT ST_AsText(ST_TransformPipeline('SRID=4939;POINT(143.0 -37.0) '::geometry,
 '+proj=pipeline
 +step +proj=unitconvert +xy_in=deg +xy_out=rad
 +step +proj=hgridshift +grids=au_icsm_GDA94_GDA2020_conformal_and_distortion.tif
 +step +proj=unitconvert +xy_in=rad +xy_out=deg')) AS gda2020_pipeline;

 gda2020_pipeline

POINT(143.0000063280214 -36.999986718287545)
(1 row)
```

## See Also

[ST\\_Transform](#), [ST\\_InverseTransformPipeline](#)

### 7.7.6 postgis\_srs\_codes

postgis\_srs\_codes — Return the list of SRS codes associated with the given authority.

#### Synopsis

setof text **postgis\_srs\_codes**(text auth\_name);

#### Description

Returns a set of all auth\_srid for the given auth\_name.

Availability: 3.4.0

Proj version 6+

#### Examples

List the first ten codes associated with the EPSG authority.

```
SELECT * FROM postgis_srs_codes('EPSG') LIMIT 10;
```

```
postgis_srs_codes

2000
20004
20005
20006
20007
20008
20009
2001
20010
20011
```

#### See Also

[postgis\\_srs](#), [postgis\\_srs\\_all](#), [postgis\\_srs\\_search](#)

### 7.7.7 postgis\_srs

postgis\_srs — Return a metadata record for the requested authority and srid.

#### Synopsis

setof record **postgis\_srs**(text auth\_name, text auth\_srid);

#### Description

Returns a metadata record for the requested auth\_srid for the given auth\_name. The record will have the auth\_name, auth\_srid, srname, srtext, proj4text, and the corners of the area of usage, point\_sw and point\_ne.

Availability: 3.4.0

Proj version 6+

---

Examples

Get the metadata for EPSG:3005.

```
SELECT * FROM postgis_srs('EPSG', '3005');

auth_name | EPSG
auth_srid | 3005
sname | NAD83 / BC Albers
srtext | PROJCS["NAD83 / BC Albers", ...]
proj4text | +proj=aea +lat_0=45 +lon_0=-126 +lat_1=50 +lat_2=58.5 +x_0=1000000 +y_0=0 +
 datum=NAD83 +units=m +no_defs +type=crs
point_sw | 0101000020E6100000E17A14AE476161C000000000000204840
point_ne | 0101000020E610000085EB51B81E855CC0E17A14AE47014E40
```

See Also

[postgis\\_srs\\_codes](#), [postgis\\_srs\\_all](#), [postgis\\_srs\\_search](#)

7.7.8 postgis\_srs\_all

postgis\_srs\_all — Return metadata records for every spatial reference system in the underlying Proj database.

Synopsis

setof record **postgis\_srs\_all**(void);

Description

Returns a set of all metadata records in the underlying Proj database. The records will have the auth\_name, auth\_srid, sname, srtext, proj4text, and the corners of the area of usage, point\_sw and point\_ne.

Availability: 3.4.0

Proj version 6+

Examples

Get the first 10 metadata records from the Proj database.

```
SELECT auth_name, auth_srid, sname FROM postgis_srs_all() LIMIT 10;
```

| auth_name | auth_srid | sname                                    |
|-----------|-----------|------------------------------------------|
| EPSG      | 2000      | Anguilla 1957 / British West Indies Grid |
| EPSG      | 20004     | Pulkovo 1995 / Gauss-Kruger zone 4       |
| EPSG      | 20005     | Pulkovo 1995 / Gauss-Kruger zone 5       |
| EPSG      | 20006     | Pulkovo 1995 / Gauss-Kruger zone 6       |
| EPSG      | 20007     | Pulkovo 1995 / Gauss-Kruger zone 7       |
| EPSG      | 20008     | Pulkovo 1995 / Gauss-Kruger zone 8       |
| EPSG      | 20009     | Pulkovo 1995 / Gauss-Kruger zone 9       |
| EPSG      | 2001      | Antigua 1943 / British West Indies Grid  |
| EPSG      | 20010     | Pulkovo 1995 / Gauss-Kruger zone 10      |
| EPSG      | 20011     | Pulkovo 1995 / Gauss-Kruger zone 11      |

See Also

[postgis\\_srs\\_codes](#), [postgis\\_srs](#), [postgis\\_srs\\_search](#)

7.7.9 postgis\_srs\_search

`postgis_srs_search` — Return metadata records for projected coordinate systems that have areas of useage that fully contain the `bounds` parameter.

Synopsis

setof record `postgis_srs_search`(geometry `bounds`, text `auth_name=EPSG`);

Description

Return a set of metadata records for projected coordinate systems that have areas of useage that fully contain the `bounds` parameter. Each record will have the `auth_name`, `auth_srid`, `sname`, `srttext`, `proj4text`, and the corners of the area of usage, `point_sw` and `point_ne`.

The search only looks for projected coordinate systems, and is intended for users to explore the possible systems that work for the extent of their data.

Availability: 3.4.0

Proj version 6+

Examples

Search for projected coordinate systems in Louisiana.

```
SELECT auth_name, auth_srid, sname,
 ST_AsText(point_sw) AS point_sw,
 ST_AsText(point_ne) AS point_ne
FROM postgis_srs_search('SRID=4326;LINESTRING(-90 30, -91 31)')
LIMIT 3;
```

| auth_name | auth_srid | sname                                | point_sw            | point_ne            |
|-----------|-----------|--------------------------------------|---------------------|---------------------|
| EPSG      | 2801      | NAD83(HARN) / Louisiana South        | POINT(-93.94 28.85) | POINT(-88.75 31.07) |
| EPSG      | 3452      | NAD83 / Louisiana South (ftUS)       | POINT(-93.94 28.85) | POINT(-88.75 31.07) |
| EPSG      | 3457      | NAD83(HARN) / Louisiana South (ftUS) | POINT(-93.94 28.85) | POINT(-88.75 31.07) |

Scan a table for max extent and find projected coordinate systems that might suit.

```
WITH ext AS (
 SELECT ST_Extent(geom) AS geom, Max(ST_SRID(geom)) AS srid
 FROM foo
)
SELECT auth_name, auth_srid, sname,
 ST_AsText(point_sw) AS point_sw,
 ST_AsText(point_ne) AS point_ne
FROM ext
CROSS JOIN postgis_srs_search(ST_SetSRID(ext.geom, ext.srid))
LIMIT 3;
```



**See Also**

[postgis\\_srs\\_codes](#), [postgis\\_srs\\_all](#), [postgis\\_srs](#)

## 7.8 Geometry Input

### 7.8.1 Well-Known Text (WKT)

#### 7.8.1.1 ST\_BdPolyFromText

**ST\_BdPolyFromText** — Construye un polígono dando una colección arbitraria de cadenas de líneas cerradas como representación "MultiLineString" de texto "Well-Known".

**Synopsis**

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

**Descripción**

Construye un polígono dando una colección arbitraria de cadenas de líneas cerradas como representación "MultiLineString" de texto "Well-Known".

**Note**

Envía un error si la cadena WKT no representa una MULTILINESTRING. Envía un error si la salida es un MULTIPOLYGON; en este caso puedes utilizar **ST\_BdMPolyFromText**, o mira **ST\_BuildArea()** para un enfoque mas específico de postgis.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2

Realizado por el módulo GEOS.

Disponibilidad: 1.1.0

**Ver también**

[ST\\_BuildArea](#), [ST\\_BdMPolyFromText](#)

#### 7.8.1.2 ST\_BdMPolyFromText

**ST\_BdMPolyFromText** — Construye un multipolígono dando una colección arbitraria de cadenas de líneas cerradas como representación "MultiLineString" de texto "Well-Known".

**Synopsis**

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

## Descripción

Construye un Polígono dando una colección arbitraria de cadenas de líneas cerradas, polígonos, "MultiLineString" en formato de texto "Well-Known".



### Note

Envía un error si el WKT no es una MULTILINESTRING. Fuerza una salida MULTIPOLYGON aunque el resultado este compuesto por un único POLYGON; puedes utilizar [ST\\_BdPolyFromText](#) si estas seguro que un único POLYGON será el resultado de la operación, o ver [ST\\_BuildArea\(\)](#) para un enfoque mas específico de postgis.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2

Realizado por el módulo GEOS.

Disponibilidad: 1.1.0

## Ver también

[ST\\_BuildArea](#), [ST\\_BdPolyFromText](#)

### 7.8.1.3 ST\_GeogFromText

ST\_GeogFromText — Devuelve un valor específico "geography" desde una representación "Well-Known Text" (WKT) o extendida.

## Synopsis

geography **ST\_GeogFromText**(text EWKT);

## Descripción

Devuelve un objeto geográfico del texto bien conocido o de la representación bien conocida extendida. Se asume SRID 4326 si no se especifica. Este es un alias para ST\_GeographyFromText. Los puntos se expresan siempre en forma latitud longitud.

## Ejemplos

```
--- converting lon lat coords to geography
ALTER TABLE sometable ADD COLUMN geog geography(POINT,4326);
UPDATE sometable SET geog = ST_GeogFromText('SRID=4326;POINT(' || lon || ' ' || lat || ')') ←
;

--- specify a geography point using EPSG:4267, NAD27
SELECT ST_AsEWKT(ST_GeogFromText('SRID=4267;POINT(-77.0092 38.889588)'));
```

## Ver también

[ST\\_AsText](#), [ST\\_GeographyFromText](#)

### 7.8.1.4 ST\_GeographyFromText

ST\_GeographyFromText — Devuelve un valor específico "geography" desde una representación "Well-Known Text" (WKT) o extendida.

**Synopsis**

geography **ST\_GeographyFromText**(text EWKT);

**Descripción**

Devuelve un objeto geográfico de la representación bien conocida de texto. Se supone SRID 4326 si no se especifica.

**Ver también**

[ST\\_GeogFromText](#), [ST\\_AsText](#)

**7.8.1.5 ST\_GeomCollFromText**

**ST\_GeomCollFromText** — Hace una colección Geometry de la colección WKT con el SRID dado. Si no se da SRID, el valor predeterminado es 0.

**Synopsis**

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

**Descripción**

Hace una colección Geometry de la representación de texto conocido (WKT) con el SRID dado. Si no se da SRID, el valor predeterminado es 0.

OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad

Devuelve null si el WKT no es una GEOMETRYCOLLECTION

**Note**

Si estas completamente seguro que todas tus geometrias WKT son colecciones, no utilices esta función. Es mas lenta que [ST\\_GeomFromText](#) ya que añade pasos de validación adicionales.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification.

**Ejemplos**

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

**Ver también**

[ST\\_GeomFromText](#), [ST\\_SRID](#)

**7.8.1.6 ST\_GeomFromEWKT**

**ST\_GeomFromEWKT** — Devuelve un valor especificado ST\_Geometry desde una representación "Extended Well-Known Text" (EWKT).

## Synopsis

geometry **ST\_GeomFromEWKT**(text EWKT);

## Descripción

Construye un objeto PostGIS ST\_Geometry desde una representación OGC "Extended Well-Known text" (EWKT).



### Note

El formato EWKT no es un estándar OGC, sino un formato específico PostGIS que incluye el identificador del sistema de referencia espacial (SRID).

Mejora: 2.0.0 se introdujeron soporte de superficies poliédricas y TIN.



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



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Ejemplos

```
SELECT ST_GeomFromEWKT('SRID=4269;LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)');
SELECT ST_GeomFromEWKT('SRID=4269;MULTILINESTRING((-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932))');

SELECT ST_GeomFromEWKT('SRID=4269;POINT(-71.064544 42.28787)');

SELECT ST_GeomFromEWKT('SRID=4269;POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 42.3903701743239,-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 42.3902909739571))');

SELECT ST_GeomFromEWKT('SRID=4269;MULTIPOLYGON(((-71.1031880899493 42.3152774590236,-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,
```

```
-71.1031880899493 42.3152774590236)),
((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873 ↵
42.315113108546)))');
```

```
--3d circular string
SELECT ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)');
```

```
--Polyhedral Surface example
SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE (
 ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
 ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
 ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
 ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
 ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
 ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)');
```

## Ver también

[ST\\_AsEWKT](#), [ST\\_GeomFromText](#)

### 7.8.1.7 ST\_GeomFromMARC21

**ST\_GeomFromMARC21** — Takes MARC21/XML geographic data as input and returns a PostGIS geometry object.

## Synopsis

geometry **ST\_GeomFromMARC21** ( text marxml );

## Descripción

This function creates a PostGIS geometry from a MARC21/XML record, which can contain a `POINT` or a `POLYGON`. In case of multiple geographic data entries in the same MARC21/XML record, a `MULTIPOINT` or `MULTIPOLYGON` will be returned. If the record contains mixed geometry types, a `GEOMETRYCOLLECTION` will be returned. It returns `NULL` if the MARC21/XML record does not contain any geographic data (datafield:034).

LOC MARC21/XML versions supported:

- [MARC21/XML 1.1](#)

Availability: 3.3.0, requires libxml2 2.6+



### Note

The MARC21/XML Coded Cartographic Mathematical Data currently does not provide any means to describe the Spatial Reference System of the encoded coordinates, so this function will always return a geometry with `SRID 0`.



### Note

Returned `POLYGON` geometries will always be clockwise oriented.

## Ejemplos

### Converting MARC21/XML geographic data containing a single POINT encoded as hddd.dddd

```
SELECT
 ST_AsText (
 ST_GeomFromMARC21 ('
 <record xmlns="http://www.loc.gov/MARC21/slim">
 <leader
>00000nz a2200000nc 4500</leader>
 <controlfield tag="001"
>040277569</controlfield>
 <datafield tag="034" ind1=" " ind2=" ">
 <subfield code="d"
>W004.500000</subfield>
 <subfield code="e"
>W004.500000</subfield>
 <subfield code="f"
>N054.250000</subfield>
 <subfield code="g"
>N054.250000</subfield>
 </datafield>
 </record>
 >'));

 st_astext

POINT(-4.5 54.25)
(1 row)
```

### Converting MARC21/XML geographic data containing a single POLYGON encoded as hdddmmss

```
SELECT
 ST_AsText (
 ST_GeomFromMARC21 ('
 <record xmlns="http://www.loc.gov/MARC21/slim">
 <leader
>01062cem a2200241 a 4500</leader>
 <controlfield tag="001"
> 84696781 </controlfield>
 <datafield tag="034" ind1="1" ind2=" ">
 <subfield code="a"
>a</subfield>
 <subfield code="b"
>50000</subfield>
 <subfield code="d"
>E0130600</subfield>
 <subfield code="e"
>E0133100</subfield>
 <subfield code="f"
>N0523900</subfield>
 <subfield code="g"
>N0522300</subfield>
 </datafield>
 </record>
 >'));

 st_astext

POLYGON((13.1 52.65,13.51666666666667 52.65,13.51666666666667 ←
52.38333333333333,13.1 52.38333333333333,13.1 52.65))
```

(1 row)

Converting MARC21/XML geographic data containing a POLYGON and a POINT:

```
SELECT
 ST_AsText (
 ST_GeomFromMARC21 ('
 <record xmlns="http://www.loc.gov/MARC21/slim">
 <datafield tag="034" ind1="1" ind2=" ">
 <subfield code="a"
>a</subfield>
 <subfield code="b"
>50000</subfield>
 <subfield code="d"
>E0130600</subfield>
 <subfield code="e"
>E0133100</subfield>
 <subfield code="f"
>N0523900</subfield>
 <subfield code="g"
>N0522300</subfield>
 </datafield>
 <datafield tag="034" ind1=" " ind2=" ">
 <subfield code="d"
>W004.500000</subfield>
 <subfield code="e"
>W004.500000</subfield>
 <subfield code="f"
>N054.250000</subfield>
 <subfield code="g"
>N054.250000</subfield>
 </datafield>
 </record>
 >'));
--
GEOMETRYCOLLECTION (POLYGON ((13.1 52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1 52.38333333333333,13.1 52.65)),POINT (-4.5 54.25))
(1 row)
```

Ver también

[ST\\_AsMARC21](#)

7.8.1.8 ST\_GeometryFromText

ST\_GeometryFromText — Devuelve un valor específico de ST\_Geometry desde una representación "Well-Known Text" (WKT). Es un alias para ST\_GeomFromText

Synopsis

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

**Descripción**

This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



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

**Ver también**

[ST\\_GeomFromText](#)

**7.8.1.9 ST\_GeomFromText**

ST\_GeomFromText — Devuelve un valor específico de ST\_Geometry desde una representación "Extended Well-Known Binary" (EWKB).

**Synopsis**

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

**Descripción**

Construye un objeto ST\_Geometry de PostGIS desde una representación OGC "Well-Known Text" (WKT).

**Note**

Hay dos variantes de la función ST\_GeomFromText. El primero no toma SRID y devuelve una geometría sin sistema de referencia espacial definido (SRID = 0). La segunda toma un SRID como segundo argumento y devuelve una geometría que incluye esta SRID como parte de sus metadatos.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2 - la opción SRID es de la suite de conformidad.



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



This method supports Circular Strings and Curves.

**Note**

While not OGC-compliant, [ST\\_MakePoint](#) is faster than ST\_GeomFromText and ST\_PointFromText. It is also easier to use for numeric coordinate values. [ST\\_Point](#) is another option similar in speed to [ST\\_MakePoint](#) and is OGC-compliant, but doesn't support anything but 2D points.

**Warning**

Cambiado: 2.0.0 En las versiones anteriores de PostGIS ST\_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') estaba permitido. Esto no está permitido ahora en PostGIS 2.0.0 para ajustarse mejor a las normas SQL/MM. Esto debería ser escrito como ST\_GeomFromText('GEOMETRYCOLLECTION EMPTY')



## Ejemplos

```

SELECT ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)');
SELECT ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)',4269);

SELECT ST_GeomFromText('MULTILINESTRING((-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932))');

SELECT ST_GeomFromText('POINT(-71.064544 42.28787)');

SELECT ST_GeomFromText('POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 42.3902909739571))');

SELECT ST_GeomFromText('MULTIPOLYGON(((-71.1031880899493 42.3152774590236,
-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,
-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,
-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,
-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,
-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,
-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,
-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,
-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,
-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,
-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,
-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,
-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,
-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,
-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,
-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,
-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,
-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,
-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,
-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,
-71.1031880899493 42.3152774590236)),
((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873 42.315113108546)))',4326);

SELECT ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)');

```

## Ver también

[ST\\_GeomFromEWKT](#), [ST\\_GeomFromWKB](#), [ST\\_SRID](#)

### 7.8.1.10 ST\_LineFromText

**ST\_LineFromText** — Hace una geometría de la representación WKT con el SRID dado. Si SRID no se da, el valor predeterminado es 0.

## Synopsis

```

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

```

## Descripción

Hace una Geometry desde un WKT con el SRID dado. Si no se da un SRID, el valor predeterminado es 0. Si el WKT pasado no es un LINESTRING, se devuelve null.



### Note

OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad



### Note

Si sabes que todas tus geometrías son LINESTRING, es mas eficiente el uso de ST\_GeomFromText. Esto llama únicamente a ST\_GeomFromText y añade validaciones adicionales que devuelven un linestring.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 7.2.8

## Ejemplos

```
SELECT ST_LineFromText('LINESTRING(1 2, 3 4)') AS aline, ST_LineFromText('POINT(1 2)') AS null_return;
aline | null_return

01020000000200000000000000000000F ... | t
```

## Ver también

[ST\\_GeomFromText](#)

### 7.8.1.11 ST\_MLineFromText

ST\_MLineFromText — Devuelve un valor especificado ST\_MultiLineString desde una representación WKT.

## Synopsis

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

## Descripción

Hace una Geometry desde el texto bien conocido (WKT) con el SRID dado. Si no se da un SRID, el valor predeterminado es 0.

OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad

Devuelve NULL si el WKT no es un MULTILINESTRING



### Note

Si estas completamente seguro que todas tus geometrias WKT son puntos, no utilices esta función. Es mas lenta que ST\_GeomFromText ya que añade algunos pasos de validación.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 9.4.4

## Ejemplos

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

## Ver también

[ST\\_GeomFromText](#)

### 7.8.1.12 ST\_MPointFromText

**ST\_MPointFromText** — Hace una geometría desde un WKT con el SRID dado. Si no se da un SRID, el valor predeterminado es 0.

## Synopsis

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

## Descripción

Hace una geometría desde un WKT con el SRID dado. Si no se da un SRID, el valor predeterminado es 0.

OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad

Devuelve NULL si el WKT no es un MULTIPUNTO



### Note

Si estas completamente seguro que todas tus geometrias WKT son puntos, no utilices esta función. Es mas lenta que ST\_GeomFromText ya que añade algunos pasos de validación.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). 3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 9.2.4

## Ejemplos

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

## Ver también

[ST\\_GeomFromText](#)

### 7.8.1.13 ST\_MPolyFromText

**ST\_MPolyFromText** — Hace una Geometría MultiPolygon desde un WKT con el SRID dado. Si no se da SRID, el valor predeterminado es 0.

## Synopsis

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

## Descripción

Hace un MultiPolygon desde un WKT con el SRID dado. Si no se da SRID, el valor predeterminado es 0.

OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad

Devuelve un error si el WKT no es un MULTIPOLYGON



### Note

Si estas completamente seguro que todas tus geometrías WKT son multipolygon, no utilices esta función. Es mas lenta que ST\_GeomFromText ya que añade algunos pasos de validación adicionales.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 9.6.4

## Ejemplos

```
SELECT ST_MPolyFromText('MULTIPOLYGON(((0 0 1,20 0 1,20 20 1,0 20 1,0 0 1),(5 5 3,5 7 3,7 7 3,7 5 3,5 5 3)))');
SELECT ST_MPolyFromText('MULTIPOLYGON(((−70.916 42.1002,−70.9468 42.0946,−70.9765 42.0872,−70.9754 42.0875,−70.9749 42.0879,−70.9752 42.0881,−70.9754 42.0891,−70.9758 42.0894,−70.9759 42.0897,−70.9759 42.0899,−70.9754 42.0902,−70.9756 42.0906,−70.9753 42.0907,−70.9753 42.0917,−70.9757 42.0924,−70.9755 42.0928,−70.9755 42.0942,−70.9751 42.0948,−70.9755 42.0953,−70.9751 42.0958,−70.9751 42.0962,−70.9759 42.0983,−70.9767 42.0987,−70.9768 42.0991,−70.9771 42.0997,−70.9771 42.1003,−70.9768 42.1005,−70.977 42.1011,−70.9766 42.1019,−70.9768 42.1026,−70.9769 42.1033,−70.9775 42.1042,−70.9773 42.1043,−70.9776 42.1043,−70.9778 42.1048,−70.9773 42.1058,−70.9774 42.1061,−70.9779 42.1065,−70.9782 42.1078,−70.9788 42.1085,−70.9798 42.1087,−70.9806 42.109,−70.9807 42.1093,−70.9806 42.1099,−70.9809 42.1109,−70.9808 42.1112,−70.9798 42.1116,−70.9792 42.1127,−70.979 42.1129,−70.9787 42.1134,−70.979 42.1139,−70.9791 42.1141,−70.9987 42.1116,−71.0022 42.1273,−70.9408 42.1513,−70.9315 42.1165,−70.916 42.1002)))',4326);
```

## Ver también

[ST\\_GeomFromText](#), [ST\\_SRID](#)

### 7.8.1.14 ST\_PointFromText

ST\_PointFromText — Crea una geometría puntual desde un WKT con el SRID dado. Si no se especifica el SRID por defecto será unknown.

## Synopsis

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

## Descripción

Construye un objeto de punto de PostGIS ST\_GEOMETRY de la representación bien conocida del texto de OGC. Si no se da SRID, se omite a desconocido (actualmente 0). Si la geometría no es una representación de punto WKT, devuelve null. Si WKT es totalmente inválido, entonces lanza un error.

**Note**

Hay 2 variantes de la función `ST_PointFromText`, la primera no toma SRID y devuelve una geometría sin sistema de referencia espacial definido. La segunda toma un id de un sistema de referencia como segundo argumento y devuelve una `ST_Geometry` que incluye este srid como parte de sus metadatos. El srid debe estar definido en la tabla `spatial_ref_sys`.

**Note**

Si estas completamente seguro que todas tus geometrias WKT son puntos, no utilices esta función. Es mas lenta que `ST_GeomFromText` ya que añade algunos pasos de validación. Si estas construyendo puntos desde coordenadas long lat y te interesan mas el rendimiento y la precisión que la conformidad con OGC, utiliza `ST_MakePoint` o el alias conforme al OGC `ST_Point`.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2 - la opción SRID es de la suite de conformidad.



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

**Ejemplos**

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

**Ver también**

[ST\\_GeomFromText](#), [ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_SRID](#)

**7.8.1.15 ST\_PolygonFromText**

`ST_PolygonFromText` — Hace una geometría desde un WKT con el SRID dado. Si no se da un SRID, el valor predeterminado es 0.

**Synopsis**

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

**Descripción**

Hace una geometría desde WKT con el SRID dado. Si no se da SRID, el valor predeterminado es 0. Devuelve null si WKT no es un polígono.

OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad

**Note**

Si estas completamente seguro que todas tus geometrías WKT son poligonos, no utilices esta función. Es mas lenta que `ST_GeomFromText` ya que añade algunos pasos de validación adicionales.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 8.3.6

## Ejemplos

```
SELECT ST_PolygonFromText('POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 ↵
 42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 ↵
 42.3902909739571))');
st_polygonfromtext

010300000001000000050000006...
```

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

t
```

## Ver también

[ST\\_GeomFromText](#)

### 7.8.1.16 ST\_WKTTToSQL

**ST\_WKTTToSQL** — Devuelve un valor específico de ST\_Geometry desde una representación "Well-Known Text" (WKT). Es un alias para ST\_GeomFromText

## Synopsis

geometry **ST\_WKTTToSQL**(text WKT);

## Descripción



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

## Ver también

[ST\\_GeomFromText](#)

## 7.8.2 Well-Known Binary (WKB)

### 7.8.2.1 ST\_GeogFromWKB

**ST\_GeogFromWKB** — Crea una instancia "geography" desde la representación de una geometría en "Well-Known Binary" (WKB) o "Extended Well-Known Binary" (EWKB).

## Synopsis

geography **ST\_GeogFromWKB**(bytea wkb);

## Descripción

La función `ST_GeogFromWKB`, toma una representación de una geometría en "Well-Known Binary" (WKB) o la versión extendida de PostGIS y crea la instancia apropiada de tipo "geography". Esta función juega el rol de "Geometry Factory" en SQL.

Si no se define un SRID, por defecto es 4326 (WGS 84 long lat).



This method supports Circular Strings and Curves.

## Ejemplos

```
--Although bytea rep contains single \, these need to be escaped when inserting into a table
SELECT ST_AsText(
ST_GeogFromWKB(E'\\001\\002\\000\\000\\000\\002\\000\\000\\000\\037\\205\\353Q
\\270~\\\\\\\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\\\\\\\300)\\\\\\\\217\\302\\365\\230
C@')
);
 st_astext

LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)
```

## Ver también

[ST\\_GeogFromText](#), [ST\\_AsBinary](#)

### 7.8.2.2 ST\_GeomFromEWKB

`ST_GeomFromEWKB` — Devuelve un valor específico de `ST_Geometry` desde una representación "Extended Well-Known Binary" (EWKB).

## Synopsis

geometry **ST\_GeomFromEWKB**(bytea EWKB);

## Descripción

Construye un objeto `ST_Geometry` de PostGIS desde un formato OGC "Extended Well-Known Binary" (EWKB).



### Note

El formato EWKB no es un estándar del OGC, sino un formato específico de PostGIS que incluye el identificador del sistema de referencia espacial (SRID)

Mejora: 2.0.0 se introdujeron soporte de superficies poliédricas y TIN.



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



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Ejemplos

Representación binaria de LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932) en NAD 83 long lat (4269).



### Note

Nota: Aunque los arrays de bits están delimitados por \ y deben tener ', necesitaremos escapar ambos con \ y " si el valor de standard\_conforming\_strings es off. Así que esto puede no ser exactamente como la representación AsEWKB.

```
SELECT ST_GeomFromEWKB(E'\001\002\000\000 \255\020\000\000\003\000\000\000\344 ←
J=
\013B\312Q\300n\303(\010\036!E@'\277E'K
\312Q\300\366{b\235*!E@\225|\354.P\312Q
\300p\231\323e1!E@');
```



### Note

En PostgreSQL 9.1 +-standard\_conforming\_strings se establece en on de forma predeterminada, donde como en versiones anteriores se estableció en off. Puede cambiar los valores predeterminados según sea necesario para una sola consulta o a nivel de base de datos o de servidor. A continuación se muestra cómo lo haría con standard\_conforming\_strings = on. En este caso nos escapamos del ' with standard ansi ', pero las barras no se escapan

```
set standard_conforming_strings = on;
SELECT ST_GeomFromEWKB('\001\002\000\000 \255\020\000\000\003\000\000\000\344J=\012\013B
\312Q\300n\303(\010\036!E@'\277E'K\012\312Q\300\366{b\235*!E@\225|\354.P\312Q\012\300 ←
p\231\323e1')
```

## Ver también

[ST\\_AsBinary](#), [ST\\_AsEWKB](#), [ST\\_GeomFromWKB](#)

### 7.8.2.3 ST\_GeomFromWKB

**ST\_GeomFromWKB** — Crea una instancia de geometría desde la representación de una geometría en "Well-Known Binary" (WKB) y un SRID opcional.

## Synopsis

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

## Descripción

La función **ST\_GeomFromWKB**, toma una representación binaria "well-known" de una geometría y un ID de un Sistema de Referencia Espacial (SRID) y crea una instancia del tipo de geometría adecuado. Esta función juega un rol de "Geometry Factory" en SQL. Es un nombre alternativo para **ST\_WKBToSQL**.

Si no se especifica SRID, el valor predeterminado es 0 (desconocido).



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.7.2 - El SRID opcional es para el paquete de conformidad



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



This method supports Circular Strings and Curves.



## Ejemplos

```
--Although bytea rep contains single \, these need to be escaped when inserting into a
table
-- unless standard_conforming_strings is set to on.
SELECT ST_AsEWKT(
ST_GeomFromWKB(E'\001\002\000\000\000\002\000\000\000\037\205\353Q
\270~\\\300\323Mb\020X\231C@\020X9\264\310~\\\300)\\\217\302\365\230
C@',4326)
);

st_asewkt

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

SELECT
 ST_AsText(
 ST_GeomFromWKB(
 ST_AsEWKB('POINT(2 5)::geometry')
)
);

st_astext

POINT(2 5)
(1 row)
```

## Ver también

[ST\\_WKBToSQL](#), [ST\\_AsBinary](#), [ST\\_GeomFromEWKB](#)

### 7.8.2.4 ST\_LineFromWKB

**ST\_LineFromWKB** — Crea un `LINESTRING` desde un `WKB` con el `SRID` dado

## Synopsis

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

## Descripción

La función `ST_GeomFromWKB`, toma una representación binaria "well-known" de una geometría y un ID de un Sistema de Referencia Espacial (`SRID`) y crea una instancia del tipo de geometría adecuado - en este caso una geometría `LINESTRING`. Esta función juega un rol de "Geometry Factory" en SQL.

Si no se especifica un `SRID`, el valor predeterminado es 0. `NULL` se devuelve si la entrada `bytea` no representa un `LINESTRING`.



### Note

OGC SPEC 3.2.6.2 - La opción `SRID` es del paquete de conformidad



### Note

Si sabes que todas tus geometrías son `LINESTRING`, es mas eficiente el uso de `ST_GeomFromWKB`. Esta función simplemente llama a `ST_GeomFromWKB` y añade validaciones adicionales y devuelve una `linestring`.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

## Ejemplos

```
SELECT ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('LINESTRING(1 2, 3 4)'))) AS aline,
 ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('POINT(1 2)'))) IS NULL AS ←
 null_return;
aline | null_return

01020000000200000000000000000000F ... | t
```

## Ver también

[ST\\_GeomFromWKB](#), [ST\\_LinestringFromWKB](#)

### 7.8.2.5 ST\_LinestringFromWKB

**ST\_LinestringFromWKB** — Crea una geometría desde un WKB con el SRID dado.

## Synopsis

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

## Descripción

La función **ST\_LinestringFromWKB**, toma una representación de una geometría en "well-known binary" y un ID de un Sistema de Referencia Espacial (SRID) y crea una instancia del tipo apropiado de geometría - en este caso, una geometría **LINESTRING**. Esta función juega un rol de "Geometry Factory" en SQL.

Si no se especifica un SRID, el valor predeterminado es 0.NULL se devuelve si la entrada *bytea* no representa una geometría **LINESTRING**. Esto es un alias para [ST\\_LineFromWKB](#).



### Note

OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad



### Note

Si sabes que todas tus geometrías son **LINESTRING**, es mas eficiente el uso de [ST\\_GeomFromWKB](#). Esta función simplemente llama a [ST\\_GeomFromWKB](#) y añade validaciones adicionales y devuelve una **LINESTRING**.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

## Ejemplos

```
SELECT
 ST_LineStringFromWKB(
 ST_AsBinary(ST_GeomFromText('LINESTRING(1 2, 3 4)'))
) AS aline,
 ST_LineStringFromWKB(
 ST_AsBinary(ST_GeomFromText('POINT(1 2)'))
) IS NULL AS null_return;
 aline | null_return

01020000000200000000000000000000F ... | t
```

## Ver también

[ST\\_GeomFromWKB](#), [ST\\_LineFromWKB](#)

### 7.8.2.6 ST\_PointFromWKB

**ST\_PointFromWKB** — Crea una geometría desde un WKB con el SRID dado.

## Synopsis

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

## Descripción

La función **ST\_PointFromWKB**, toma una representación binaria "well-known" de una geometría y un ID de un Sistema de Referencia Espacial (SRID) y crea una instancia del tipo de geometría adecuado - en este caso una geometría **POINT**. Esta función juega un rol de "Geometry Factory" en SQL.

Si no se especifica un SRID, el valor predeterminado es 0. **NULL** se devuelve si la entrada **bytea** no representa una geometría de **POINT**.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.7.2



This method implements the SQL/MM specification. SQL-MM 3: 6.1.9



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



This method supports Circular Strings and Curves.

## Ejemplos

```
SELECT
 ST_AsText (
 ST_PointFromWKB (
 ST_AsEWKB ('POINT(2 5) '::geometry)
)
);
 st_astext

POINT(2 5)
(1 row)
```

```

SELECT
 ST_AsText (
 ST_PointFromWKB (
 ST_AsEWKB ('LINESTRING(2 5, 2 6)::geometry')
)
);
 st_astext

(1 row)

```

**Ver también**

[ST\\_GeomFromWKB](#), [ST\\_LineFromWKB](#)

**7.8.2.7 ST\_WKBToSQL**

**ST\_WKBToSQL** — Devuelve un valor específico de ST\_Geometry desde una representación "Well-Known Binary" (WKB). Es un alias para ST\_GeomFromWKB que no toma srid

**Synopsis**

geometry **ST\_WKBToSQL**(bytea WKB);

**Descripción**

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

**Ver también**

[ST\\_GeomFromWKB](#)

**7.8.3 Other Formats****7.8.3.1 ST\_Box2dFromGeoHash**

**ST\_Box2dFromGeoHash** — Devuelve un BOX2D de una cadena de GeoHash.

**Synopsis**

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

**Descripción**

Devuelve un BOX2D de una cadena de GeoHash.

If no `precision` is specified **ST\_Box2dFromGeoHash** returns a BOX2D based on full precision of the input GeoHash string.

Si es especificada la `precisión` **ST\_Box2dFromGeoHash** utilizará muchos caracteres del GeoHash para crear el BOX2D. Los valores de precisión más bajos resultan en BOX2Ds más grandes y los valores más grandes aumentan la precisión.

Disponibilidad: 2.1.0

## Ejemplos

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0');

 st_geomfromgeohash

BOX(-115.172816 36.114646,-115.172816 36.114646)

SELECT ST_Box2dFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0', 0);

 st_box2dfromgeohash

BOX(-180 -90,180 90)

SELECT ST_Box2dFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0', 10);

 st_box2dfromgeohash

BOX(-115.17282128334 36.1146408319473,-115.172810554504 36.1146461963654)
```

## Ver también

[ST\\_GeoHash](#), [ST\\_GeomFromGeoHash](#), [ST\\_PointFromGeoHash](#)

### 7.8.3.2 ST\_GeomFromGeoHash

ST\_GeomFromGeoHash — Devuelve una geometría de una cadena de GeoHash.

## Synopsis

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

## Descripción

Devuelve una geometría de una cadena de GeoHash. La geometría será un polígono que representa los límites de GeoHash.

Si no se especifica ninguna *precisión*, ST\_GeomFromGeoHash devuelve un polígono basándose en la precisión completa de la cadena de GeoHash de entrada.

Si se especifica la *precisión*, ST\_GeomFromGeoHash utilizará muchos caracteres del GeoHash para crear el polígono.

Disponibilidad: 2.1.0

## Ejemplos

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0'));

 st_astext

POLYGON((-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816 ↵
36.114646,-115.172816 36.114646))

SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0', 4));

 st_astext

POLYGON((-115.3125 36.03515625,-115.3125 36.2109375,-114.9609375 36.2109375,-114.9609375 ↵
36.03515625,-115.3125 36.03515625))
```

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0', 10));
```

st\_astext ↩

```
POLYGON((-115.17282128334 36.1146408319473,-115.17282128334 ↩
 36.1146461963654,-115.172810554504 36.1146461963654,-115.172810554504 ↩
 36.1146408319473,-115.17282128334 36.1146408319473))
```

## Ver también

[ST\\_GeoHash](#), [ST\\_Box2dFromGeoHash](#), [ST\\_PointFromGeoHash](#)

### 7.8.3.3 ST\_GeomFromGML

**ST\_GeomFromGML** — Toma una representación GML como entrada de una geometría y extrae un objeto geométrico PostGIS

#### Synopsis

```
geometry ST_GeomFromGML(text geomgml);
geometry ST_GeomFromGML(text geomgml, integer srid);
```

#### Descripción

Construye un objeto **ST\_Geometry** de PostGIS desde una representación OGC GML.

**ST\_GeomFromGML** funciona solamente para fragmentos geométricos GML. Lanza un error si intentas utilizar un documento GML completo.

Versiones OGC GML soportadas:

- GML 3.2.1 Namespace
- GML 3.1.1 Simple Features profile SF-2 (con GML 3.1.0 y 3.0.0 compatibilidad para versiones anteriores)
- GML 2.1.2

OGC GML standards, cf: <http://www.opengeospatial.org/standards/gml>:

Disponibilidad: 1.5, requiere libxml2 1.6+

Mejora: 2.0.0 se introdujeron soporte de superficies poliédricas y TIN.

Mejorada: 2.0.0 se agregó el parámetro por defecto opcional srid.



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



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

GML permite dimensiones mixtas (2D y 3D dentro de la misma MultiGeometry, por ejemplo). Como las geometrías PostGIS no lo hacen, **ST\_GeomFromGML** convierte todas las geometrías a 2D si se encuentra una dimensión Z que falta.

GML soporta SRS diferentes en la misma MultiGeometry. Como las geometrías de PostGIS no lo hacen, **ST\_GeomFromGML**, en este caso, reproyecta todas las subgeometrías al SRS del nodo padre. Si no esta disponible el atributo srsName en el nodo padre del GML, la función lanza un error.

La función **ST\_GeomFromGML** no es muy estricta con los namespaces explícitos de un GML. Puedes evitar mencionarlos explícitamente para usos comunes. Pero lo necesitas si deseas utilizar la función **XLink** dentro del GML.

**Note**

La función ST\_GeomFromGML no soporta geometrias curvas SQL/MM.

**Ejemplos - Una geometría simple con srsName**

```
SELECT ST_GeomFromGML('
 <gml:LineString srsName="EPSG:4269">
 <gml:coordinates>
 -71.16028,42.258729 -71.160837,42.259112 ↵
 -71.161143,42.25932
 </gml:coordinates>
 </gml:LineString
>');
```

**Ejemplos - uso de XLink**

```
SELECT ST_GeomFromGML('
 <gml:LineString xmlns:gml="http://www.opengis.net/gml"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 srsName="urn:ogc:def:crs:EPSG::4269">
 <gml:pointProperty>
 <gml:Point gml:id="p1"
><gml:pos
>42.258729 -71.16028</gml:pos
></gml:Point>
 </gml:pointProperty>
 <gml:pos
>42.259112 -71.160837</gml:pos>
 <gml:pointProperty>
 <gml:Point xlink:type="simple" xlink:href="#p1"/>
 </gml:pointProperty>
 </gml:LineString
>'););
```

**Ejemplos - Superficie polihédrica**

```
SELECT ST_AsEWKT(ST_GeomFromGML('
<gml:PolyhedralSurface>
<gml:polygonPatches>
 <gml:PolygonPatch>
 <gml:exterior>
 <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList
></gml:LinearRing>
 </gml:exterior>
 </gml:PolygonPatch>
 <gml:PolygonPatch>
 <gml:exterior>
 <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList
></gml:LinearRing>
 </gml:exterior>
 </gml:PolygonPatch>
```

```

 <gml:PolygonPatch>
 <gml:exterior>
 <gml:LinearRing>
><gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList
></gml:LinearRing>
 </gml:exterior>
 </gml:PolygonPatch>
 <gml:PolygonPatch>
 <gml:exterior>
 <gml:LinearRing>
><gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList
></gml:LinearRing>
 </gml:exterior>
 </gml:PolygonPatch>
 <gml:PolygonPatch>
 <gml:exterior>
 <gml:LinearRing>
><gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 0 0 1 0</gml:posList
></gml:LinearRing>
 </gml:exterior>
 </gml:PolygonPatch>
 <gml:PolygonPatch>
 <gml:exterior>
 <gml:LinearRing>
><gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList
></gml:LinearRing>
 </gml:exterior>
 </gml:PolygonPatch>
</gml:polygonPatches>
</gml:PolyhedralSurface
>'))';

-- result --
POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))

```

## Ver también

Section [2.2.3](#), [ST\\_AsGML](#), [ST\\_GMLToSQL](#)

### 7.8.3.4 ST\_GeomFromGeoJSON

**ST\_GeomFromGeoJSON** — Toma como entrada una representación geojson de una geometría y devuelve un objeto geométrico PostGIS

## Synopsis

```

geometry ST_GeomFromGeoJSON(text geomjson);
geometry ST_GeomFromGeoJSON(json geomjson);
geometry ST_GeomFromGeoJSON(jsonb geomjson);

```



**Descripción**

Construye un objeto geométrico PostGIS desde una representación GeoJSON.

ST\_GeomFromGeoJSON solo funciona con fragmentos geométricos JSON. Devolverá un error si intentas utilizar un documento JSON completo.

Enhanced: 3.0.0 parsed geometry defaults to SRID=4326 if not specified otherwise.

Enhanced: 2.5.0 can now accept json and jsonb as inputs.

Disponibilidad: 2.0.0 necesita de - JSON-C >= 0.9

**Note**

Si no tienes activado el soporte de JSON-C, tendrás un mensaje error en vez de ver la salida. Para activar el soporte JSON-C, ejecuta `configure --with-jsondir=/path/to/json-c`. Para mas detalles ve a Section [2.2.3](#).



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

**Ejemplos**

```
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[-48.23456,20.12345]}')) ←
 As wkt;

POINT(-48.23456 20.12345)
```

```
-- a 3D linestring
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"LineString","coordinates ←
 ":[[1,2,3], [4,5,6], [7,8,9]]}')) As wkt;
wkt

LINESTRING(1 2,4 5,7 8)
```

**Ver también**

[ST\\_AsText](#), [ST\\_AsGeoJSON](#), Section [2.2.3](#)

**7.8.3.5 ST\_GeomFromKML**

ST\_GeomFromKML — Toma una representación de una geometría KML de entrada y devuelve un objeto geométrico PostGIS

**Synopsis**

geometry **ST\_GeomFromKML**(text geomkml);

**Descripción**

Construye un objeto ST\_Geometry de PostGIS desde una representación OGC KML.

ST\_GeomFromKML solo funciona con fragmentos geométricos KML. Devuelve un error si intentas utilizar un documento KML completo.

Versiones soportadas OGC KML:

- KML 2.2.0 Namespace

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

Availability: 1.5, requires libxml2 2.6+



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



#### Note

ST\_GeomFromKML no soporta geometrías curvas SQL/MM.

### Ejemplos - Una geometría simple con srsName

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

### Ver también

Section 2.2.3, [ST\\_AsKML](#)

### 7.8.3.6 ST\_GeomFromTWKB

ST\_GeomFromTWKB — Crea una instancia de geometría de una representación geométrica TWKB ("Tiny Well-Known Binary").

### Synopsis

geometry **ST\_GeomFromTWKB**(bytea twkb);

### Descripción

La función ST\_GeomFromTWKB toma un TWKB ("Tiny Well-Known Binary") a una representación geométrica (WKB) y crea una instancia apropiada de un tipo de geometría.

### Ejemplos

```
SELECT ST_AsText(ST_GeomFromTWKB(ST_AsTWKB('LINESTRING(126 34, 127 35)::geometry')));

 st_astext

LINESTRING(126 34, 127 35)
(1 row)

SELECT ST_AsEWKT(
 ST_GeomFromTWKB(E'\x620002f7f40dbce4040105')
);
```

```
----- st_asewkt -----
```

```
LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)
```

#### Ver también

[ST\\_AsTWKB](#)

#### 7.8.3.7 ST\_GMLToSQL

**ST\_GMLToSQL** — Devuelve un valor específico ST\_Geometry desde una representación GML. Esto es un alias de [ST\\_GeomFromGML](#).

#### Synopsis

```
geometry ST_GMLToSQL(text geomgml);
geometry ST_GMLToSQL(text geomgml, integer srid);
```

#### Descripción



This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (excepto para soporte de curvas).

Disponibilidad: 1.5, requiere libxml2 1.6+

Mejora: 2.0.0 se introdujeron soporte de superficies poliédricas y TIN.

Mejorada: 2.0.0 se agregó el parámetro por defecto opcional srid.

#### Ver también

Section [2.2.3](#), [ST\\_GeomFromGML](#), [ST\\_AsGML](#)

#### 7.8.3.8 ST\_LineFromEncodedPolyline

**ST\_LineFromEncodedPolyline** — Crea un LineString desde una polilínea codificada.

#### Synopsis

```
geometry ST_LineFromEncodedPolyline(text polyline, integer precision=5);
```

#### Descripción

Crea un LineString desde una cadena de polilínea codificada.

Optional `precision` specifies how many decimal places will be preserved in Encoded Polyline. Value should be the same on encoding and decoding, or coordinates will be incorrect.

Ver <http://developers.google.com/maps/documentation/utilities/polylinealgorithm>

Disponibilidad: 2.2.0

## Ejemplos

```
-- Create a line string from a polyline
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@'));
-- result --
SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)

-- Select different precision that was used for polyline encoding
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@',6));
-- result --
SRID=4326;LINESTRING(-12.02 3.85,-12.095 4.07,-12.6453 4.3252)
```

## Ver también

[ST\\_AsEncodedPolyline](#)

### 7.8.3.9 ST\_PointFromGeoHash

**ST\_PointFromGeoHash** — Devuelve un punto de una cadena de GeoHash.

## Synopsis

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

## Descripción

Devuelve un punto de una cadena de GeoHash. El punto representa el punto central del GeoHash.

Si no se especifica ninguna *precisión*, **ST\_PointFromGeoHash** devuelve un punto basándose en la precisión completa de la cadena de GeoHash de entrada.

Si *precision* es especificado **ST\_PointFromGeoHash** utilizará muchos caracteres de GeoHash para crear el punto.

Disponibilidad: 2.1.0

## Ejemplos

```
SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0'));
 st_astext

POINT(-115.172816 36.114646)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0', 4));
 st_astext

POINT(-115.13671875 36.123046875)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0', 10));
 st_astext

POINT(-115.172815918922 36.1146435141563)
```

## Ver también

[ST\\_GeoHash](#), [ST\\_Box2dFromGeoHash](#), [ST\\_GeomFromGeoHash](#)

### 7.8.3.10 ST\_FromFlatGeobufToTable

ST\_FromFlatGeobufToTable — Creates a table based on the structure of FlatGeobuf data.

#### Synopsis

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

#### Descripción

Creates a table based on the structure of FlatGeobuf data. (<http://flatgeobuf.org>).

`schema` Schema name.

`table` Table name.

`data` Input FlatGeobuf data.

Availability: 3.2.0

### 7.8.3.11 ST\_FromFlatGeobuf

ST\_FromFlatGeobuf — Reads FlatGeobuf data.

#### Synopsis

setof anyelement **ST\_FromFlatGeobuf**(anyelement Table reference, bytea FlatGeobuf input data);

#### Descripción

Reads FlatGeobuf data (<http://flatgeobuf.org>). NOTE: PostgreSQL bytea cannot exceed 1GB.

`tabletype` reference to a table type.

`data` input FlatGeobuf data.

Availability: 3.2.0

## 7.9 Geometry Output

### 7.9.1 Well-Known Text (WKT)

#### 7.9.1.1 ST\_AsEWKT

ST\_AsEWKT — Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.

#### Synopsis

text **ST\_AsEWKT**(geometry g1);

text **ST\_AsEWKT**(geometry g1, integer maxdecimaldigits=15);

text **ST\_AsEWKT**(geography g1);

text **ST\_AsEWKT**(geography g1, integer maxdecimaldigits=15);

---

## Descripción

Returns the Well-Known Text representation of the geometry prefixed with the SRID. The optional *maxdecimaldigits* argument may be used to reduce the maximum number of decimal digits after floating point used in output (defaults to 15).

To perform the inverse conversion of EWKT representation to PostGIS geometry use [ST\\_GeomFromEWKT](#).



### Warning

Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use [ST\\_ReducePrecision](#) with a suitable gridsize first.



### Note

The WKT spec does not include the SRID. To get the OGC WKT format use [ST\\_AsText](#).



### Warning

WKT format does not maintain precision so to prevent floating truncation, use [ST\\_AsBinary](#) or [ST\\_AsEWKB](#) format for transport.

Enhanced: 3.1.0 support for optional precision parameter.

Enhanced: 2.0.0 support for Geography, Polyhedral surfaces, Triangles and TIN was introduced.



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



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Ejemplos

```
SELECT ST_AsEWKT('0103000020E61000000100000005000000000000
 0000000000000000000000000000000000000000
 F03F000000000000F03F000000000000F03F000000000000F03
 F0000000000000000000000000000000000000000'::geometry);

 st_asewkt

SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
(1 row)

SELECT ST_AsEWKT('01080000800300000000000000000060 ↵
 E30A4100000000785C0241000000000000F03F0000000018
E20A4100000000485F024100000000000000400000000018
E20A4100000000305C024100000000000000840')

--st_asewkt---
CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)
```

**Ver también**

[ST\\_AsBinary](#), [ST\\_AsEWKB](#), [ST\\_AsText](#), [ST\\_GeomFromEWKT](#)

**7.9.1.2 ST\_AsText**

**ST\_AsText** — Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.

**Synopsis**

```
text ST_AsText(geometry g1);
text ST_AsText(geometry g1, integer maxdecimaldigits = 15);
text ST_AsText(geography g1);
text ST_AsText(geography g1, integer maxdecimaldigits = 15);
```

**Descripción**

Returns the OGC **Well-Known Text** (WKT) representation of the geometry/geography. The optional *maxdecimaldigits* argument may be used to limit the number of digits after the decimal point in output ordinates (defaults to 15).

To perform the inverse conversion of WKT representation to PostGIS geometry use [ST\\_GeomFromText](#).

**Note**

The standard OGC WKT representation does not include the SRID. To include the SRID as part of the output representation, use the non-standard PostGIS function [ST\\_AsEWKT](#)

**Warning**

The textual representation of numbers in WKT may not maintain full floating-point precision. To ensure full accuracy for data storage or transport it is best to use **Well-Known Binary** (WKB) format (see [ST\\_AsBinary](#) and *maxdecimaldigits*).

**Warning**

Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use [ST\\_ReducePrecision](#) with a suitable gridsize first.

Availability: 1.5 - support for geography was introduced.

Enhanced: 2.5 - optional parameter precision introduced.



This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**, s2.1.1.1



This method implements the SQL/MM specification, SQL-MM 3: 5.1.25



This method supports Circular Strings and Curves.







### Note

The default behavior in PostgreSQL 9.0 has been changed to output bytea in hex encoding. If your GUI tools require the old behavior, then SET bytea\_output='escape' in your database.

Mejorado: 2.0.0 soporte para superficies poliédricas, triángulos y TIN fue introducida.

Enhanced: 2.0.0 support for higher coordinate dimensions was introduced.

Enhanced: 2.0.0 support for specifying endian with geography was introduced.

Availability: 1.5.0 geography support was introduced.

Changed: 2.0.0 Inputs to this function can not be unknown -- must be geometry. Constructs such as `ST_AsBinary('POINT(1 2)')` are no longer valid and you will get an `n_st_asbinary(unknown) is not unique` error. Code like that needs to be changed to `ST_AsBinary('POINT(1 2)'::geometry);`. If that is not possible, then install `legacy.sql`.



This method implements the **OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1**



This method implements the SQL/MM specification. SQL-MM 3: 5.1.37



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



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

## Ejemplos

[illegible][illegible]

### Ver también

ST\_GeomFromWKB, ST\_AsEWKB, ST\_AsTWKB, ST\_AsText,

#### 7.9.2.2 ST AsEWKB

**ST\_AsEWKB** — Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.



## Synopsis

```
text ST_AsHEXEWKB(geometry g1, text NDRorXDR);
text ST_AsHEXEWKB(geometry g1);
```

## Descripción

Returns a Geometry in HEXEWKB format (as text) using either little-endian (NDR) or big-endian (XDR) encoding. If no encoding is specified, then NDR is used.



### Note

Disponibilidad: 1.2.2



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



This method supports Circular Strings and Curves.

## Ejemplos

```
SELECT ST_AsHEXEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
 which gives same answer as

SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326)::text;

st_ashexewkb

0103000020E6100000010000000500
000000000000000000000000000000
000000000000000000000000000000F03F
00000000000000F03F000000000000F03F000000000000F03
F000000000000000000000000000000000000000000000000
```

## 7.9.3 Other Formats

### 7.9.3.1 ST\_AsEncodedPolyline

ST\_AsEncodedPolyline — Returns an Encoded Polyline from a LineString geometry.

## Synopsis

```
text ST_AsEncodedPolyline(geometry geom, integer precision=5);
```

## Descripción

Returns the geometry as an Encoded Polyline. This format is used by Google Maps with precision=5 and by Open Source Routing Machine with precision=5 and 6.

Optional `precision` specifies how many decimal places will be preserved in Encoded Polyline. Value should be the same on encoding and decoding, or coordinates will be incorrect.

Disponibilidad: 2.2.0

## Ejemplos

### Basic

```
SELECT ST_AsEncodedPolyline(GeomFromEWKT('SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)'));
--result--
|_p~iF~ps|U_ulLnnqC_mqNvxq`@
```

Use in conjunction with geography linestring and geography segmentize, and put on google maps

```
-- the SQL for Boston to San Francisco, segments every 100 KM
SELECT ST_AsEncodedPolyline(
 ST_Segmentize(
 ST_GeogFromText('LINESTRING(-71.0519 42.4935,-122.4483 37.64)'),
 100000)::geometry) As encodedFlightPath;
```

javascript will look something like this where \$ variable you replace with query result

```
<script type="text/javascript" src="http://maps.googleapis.com/maps/api/js?libraries=geometry">
</script>
<script type="text/javascript">
 flightPath = new google.maps.Polyline({
 path: google.maps.geometry.encoding.decodePath("$encodedFlightPath"),
 map: map,
 strokeColor: '#0000CC',
 strokeOpacity: 1.0,
 strokeWeight: 4
 });
</script>
```

### Ver también

[ST\\_LineFromEncodedPolyline](#), [ST\\_Segmentize](#)

### 7.9.3.2 ST\_AsFlatGeobuf

**ST\_AsFlatGeobuf** — Return a FlatGeobuf representation of a set of rows.

#### Synopsis

```
bytea ST_AsFlatGeobuf(anyelement set row);
bytea ST_AsFlatGeobuf(anyelement row, bool index);
bytea ST_AsFlatGeobuf(anyelement row, bool index, text geom_name);
```

#### Descripción

Return a FlatGeobuf representation (<http://flatgeobuf.org>) of a set of rows corresponding to a FeatureCollection. NOTE: Post-greSQL bytea cannot exceed 1GB.

*row* row data with at least a geometry column.

*index* toggle spatial index creation. Default is false.

*geom\_name* is the name of the geometry column in the row data. If NULL it will default to the first found geometry column.

Availability: 3.2.0

### 7.9.3.3 ST\_AsGeobuf

ST\_AsGeobuf — Return a Geobuf representation of a set of rows.

#### Synopsis

bytea **ST\_AsGeobuf**(anyelement set row);  
 bytea **ST\_AsGeobuf**(anyelement row, text geom\_name);

#### Descripción

Return a Geobuf representation (<https://github.com/mapbox/geobuf>) of a set of rows corresponding to a FeatureCollection. Every input geometry is analyzed to determine maximum precision for optimal storage. Note that Geobuf in its current form cannot be streamed so the full output will be assembled in memory.

row row data with at least a geometry column.

geom\_name is the name of the geometry column in the row data. If NULL it will default to the first found geometry column.

Availability: 2.4.0

#### Ejemplos

```
SELECT encode(ST_AsGeobuf(q, 'geom'), 'base64')
 FROM (SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))') AS geom) AS q;
st_asgeobuf

GAAiEAoOCgwIBBoIAAAAAgIAAAE=
```

### 7.9.3.4 ST\_AsGeoJSON

ST\_AsGeoJSON — Return a geometry as a GeoJSON element.

#### Synopsis

text **ST\_AsGeoJSON**(record feature, text geomcolumnname, integer maxdecimaldigits=9, boolean pretty\_bool=false);  
 text **ST\_AsGeoJSON**(geometry geom, integer maxdecimaldigits=9, integer options=8);  
 text **ST\_AsGeoJSON**(geography geog, integer maxdecimaldigits=9, integer options=0);

#### Descripción

Returns a geometry as a GeoJSON "geometry", or a row as a GeoJSON "feature". (See the [GeoJSON specifications RFC 7946](#)). 2D and 3D Geometries are both supported. GeoJSON only support SFS 1.1 geometry types (no curve support for example).

The `maxdecimaldigits` argument may be used to reduce the maximum number of decimal places used in output (defaults to 9). If you are using EPSG:4326 and are outputting the geometry only for display, `maxdecimaldigits=6` can be a good choice for many maps.



#### Warning

Using the `maxdecimaldigits` parameter can cause output geometry to become invalid. To avoid this use **ST\_ReducePrecision** with a suitable gridsize first.

The `options` argument can be used to add BBOX or CRS in GeoJSON output:

- 0: means no option
- 1: GeoJSON BBOX
- 2: GeoJSON Short CRS (e.g EPSG:4326)
- 4: GeoJSON Long CRS (e.g `urn:ogc:def:crs:EPSG::4326`)
- 8: GeoJSON Short CRS if not EPSG:4326 (default)

The GeoJSON specification states that polygons are oriented using the Right-Hand Rule, and some clients require this orientation. This can be ensured by using `ST_ForcePolygonCCW`. The specification also requires that geometry be in the WGS84 coordinate system (SRID = 4326). If necessary geometry can be projected into WGS84 using `ST_Transform`: `ST_Transform(geom, 4326)`.

GeoJSON can be tested and viewed online at [geojson.io](http://geojson.io) and [geojsonlint.com](http://geojsonlint.com). It is widely supported by web mapping frameworks:

- [OpenLayers GeoJSON Example](#)
- [Leaflet GeoJSON Example](#)
- [Mapbox GL GeoJSON Example](#)

Disponibilidad: 1.3.4

Availability: 1.5.0 geography support was introduced.

Changed: 2.0.0 support default args and named args.

Changed: 3.0.0 support records as input

Changed: 3.0.0 output SRID if not EPSG:4326.



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

## Ejemplos

Generate a FeatureCollection:

```
SELECT json_build_object(
 'type', 'FeatureCollection',
 'features', json_agg(ST_AsGeoJSON(t.*)::json)
)
FROM (VALUES (1, 'one', 'POINT(1 1)::geometry),
 (2, 'two', 'POINT(2 2)'),
 (3, 'three', 'POINT(3 3)')
) as t(id, name, geom);
```

```
{"type" : "FeatureCollection", "features" : [{"type": "Feature", "geometry": {"type": "Point", "coordinates": [1,1]}, "properties": {"id": 1, "name": "one"}}, {"type": "Feature", "geometry": {"type": "Point", "coordinates": [2,2]}, "properties": {"id": 2, "name": "two"}}, {"type": "Feature", "geometry": {"type": "Point", "coordinates": [3,3]}, "properties": {"id": 3, "name": "three"}}]}
```

Generate a Feature:

```
SELECT ST_AsGeoJSON(t.*)
FROM (VALUES (1, 'one', 'POINT(1 1)::geometry)) AS t(id, name, geom);
```

```
st_asgeojson
```

---

```
{ "type": "Feature", "geometry": { "type": "Point", "coordinates": [1,1] }, "properties": { "id": 1, "name": "one" } }
```

An alternate way to generate Features with an id property is to use JSONB functions and operators:

```
SELECT jsonb_build_object(
 'type', 'Feature',
 'id', id,
 'geometry', ST_AsGeoJSON(geom)::jsonb,
 'properties', to_jsonb(t.*) - 'id' - 'geom'
) AS json
FROM (VALUES (1, 'one', 'POINT(1 1)::geometry)) AS t(id, name, geom);
```

```
json
```

---

```
{ "id": 1, "type": "Feature", "geometry": { "type": "Point", "coordinates": [1, 1] }, "properties": { "name": "one" } }
```

Don't forget to transform your data to WGS84 longitude, latitude to conform with the GeoJSON specification:

```
SELECT ST_AsGeoJSON(ST_Transform(geom,4326)) from fe_edges limit 1;
```

```
st_asgeojson
```

---

```
{ "type": "MultiLineString", "coordinates": [[[-89.734634999999997, 31.492072000000000], [-89.734955999999997, 31.492237999999997]]] }
```

3D geometries are supported:

```
SELECT ST_AsGeoJSON('LINESTRING(1 2 3, 4 5 6)');
```

```
{ "type": "LineString", "coordinates": [[1,2,3], [4,5,6]] }
```

**Ver también**

[ST\\_GeomFromGeoJSON](#), [ST\\_ForcePolygonCCW](#), [ST\\_Transform](#)

### 7.9.3.5 ST\_AsGML

**ST\_AsGML** — Return the geometry as a GML version 2 or 3 element.

#### Synopsis

```
text ST_AsGML(geometry geom, integer maxdecimaldigits=15, integer options=0);
text ST_AsGML(geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);
text ST_AsGML(integer version, geometry geom, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);
text ST_AsGML(integer version, geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);
```

## Descripción

Return the geometry as a Geography Markup Language (GML) element. The version parameter, if specified, may be either 2 or 3. If no version parameter is specified then the default is assumed to be 2. The `maxdecimaldigits` argument may be used to reduce the maximum number of decimal places used in output (defaults to 15).



### Warning

Using the `maxdecimaldigits` parameter can cause output geometry to become invalid. To avoid this use `ST_ReducePrecision` with a suitable gridsize first.

---

GML 2 refer to 2.1.2 version, GML 3 to 3.1.1 version

The 'options' argument is a bitfield. It could be used to define CRS output type in GML output, and to declare data as lat/lon:

- 0: GML Short CRS (e.g EPSG:4326), default value
- 1: GML Long CRS (e.g urn:ogc:def:crs:EPSG::4326)
- 2: For GML 3 only, remove srsDimension attribute from output.
- 4: For GML 3 only, use <LineString> rather than <Curve> tag for lines.
- 16: Declare that datas are lat/lon (e.g srid=4326). Default is to assume that data are planars. This option is useful for GML 3.1.1 output only, related to axis order. So if you set it, it will swap the coordinates so order is lat lon instead of database lon lat.
- 32: Output the box of the geometry (envelope).

The 'namespace prefix' argument may be used to specify a custom namespace prefix or no prefix (if empty). If null or omitted 'gml' prefix is used

Disponibilidad: 1.3.2

Availability: 1.5.0 geography support was introduced.

Enhanced: 2.0.0 prefix support was introduced. Option 4 for GML3 was introduced to allow using LineString instead of Curve tag for lines. GML3 Support for Polyhedral surfaces and TINS was introduced. Option 32 was introduced to output the box.

Changed: 2.0.0 use default named args

Enhanced: 2.1.0 id support was introduced, for GML 3.



### Note

Only version 3+ of ST\_AsGML supports Polyhedral Surfaces and TINS.

---



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 17.2



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



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

---



**Ejemplos: Versión 2**

```
SELECT ST_AsGML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
 st_asgml

 <gml:Polygon srsName="EPSG:4326"
><gml:outerBoundaryIs
><gml:LinearRing
><gml:coordinates
>0,0 0,1 1,1 1,0 0,0</gml:coordinates
></gml:LinearRing
></gml:outerBoundaryIs
></gml:Polygon
>
```

**Ejemplos: Versión 3**

```
-- Flip coordinates and output extended EPSG (16 | 1)--
SELECT ST_AsGML(3, ST_GeomFromText('POINT(5.234234233242 6.34534534534)',4326), 5, 17);
 st_asgml

 <gml:Point srsName="urn:ogc:def:crs:EPSG::4326"
><gml:pos
>6.34535 5.23423</gml:pos
></gml:Point
>
```

```
-- Output the envelope (32) --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 32);
 st_asgml

 <gml:Envelope srsName="EPSG:4326">
 <gml:lowerCorner
>1 2</gml:lowerCorner>
 <gml:upperCorner
>10 20</gml:upperCorner>
 </gml:Envelope
>
```

```
-- Output the envelope (32) , reverse (lat lon instead of lon lat) (16), long srs (1)= 32 | ←
16 | 1 = 49 --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 49);
 st_asgml

<gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
 <gml:lowerCorner
>2 1</gml:lowerCorner>
 <gml:upperCorner
>20 10</gml:upperCorner>
</gml:Envelope
>
```

```
-- Polyhedral Example --
SELECT ST_AsGML(3, ST_GeomFromEWKT('POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0) ←
),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))''));
 st_asgml
```

```

<gml:PolyhedralSurface>
<gml:polygonPatches>
 <gml:PolygonPatch>
 <gml:exterior>
 <gml:LinearRing>
 <gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList>
 </gml:LinearRing>
 </gml:exterior>
 </gml:PolygonPatch>
 <gml:PolygonPatch>
 <gml:exterior>
 <gml:LinearRing>
 <gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList>
 </gml:LinearRing>
 </gml:exterior>
 </gml:PolygonPatch>
 <gml:PolygonPatch>
 <gml:exterior>
 <gml:LinearRing>
 <gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList>
 </gml:LinearRing>
 </gml:exterior>
 </gml:PolygonPatch>
 <gml:PolygonPatch>
 <gml:exterior>
 <gml:LinearRing>
 <gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList>
 </gml:LinearRing>
 </gml:exterior>
 </gml:PolygonPatch>
 <gml:PolygonPatch>
 <gml:exterior>
 <gml:LinearRing>
 <gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 0 0 1 0</gml:posList>
 </gml:LinearRing>
 </gml:exterior>
 </gml:PolygonPatch>
 <gml:PolygonPatch>
 <gml:exterior>
 <gml:LinearRing>
 <gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>
 </gml:LinearRing>
 </gml:exterior>
 </gml:PolygonPatch>
</gml:polygonPatches>
</gml:PolyhedralSurface>
>

```

**Ver también**

[ST\\_GeomFromGML](#)

### 7.9.3.6 ST\_AsKML

ST\_AsKML — Return the geometry as a KML element.

#### Synopsis

text **ST\_AsKML**(geometry geom, integer maxdecimaldigits=15, text nprefix=NULL);

text **ST\_AsKML**(geography geog, integer maxdecimaldigits=15, text nprefix=NULL);

#### Descripción

Return the geometry as a Keyhole Markup Language (KML) element. default maximum number of decimal places is 15, default namespace is no prefix.



#### Warning

Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use **ST\_ReducePrecision** with a suitable gridsize first.



#### Note

Requiere que PostGIS sea compilado con soporte de Proj. Utilice **PostGIS\_Full\_Version** para confirmar que ha compilado el soporte de proyectos.



#### Note

Availability: 1.2.2 - later variants that include version param came in 1.3.2



#### Note

Enhanced: 2.0.0 - Add prefix namespace, use default and named args



#### Note

Changed: 3.0.0 - Removed the "versioned" variant signature



#### Note

AsKML output will not work with geometries that do not have an SRID



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

## Ejemplos

```
SELECT ST_AsKML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));

 st_askml
 -
 <Polygon
><outerBoundaryIs
><LinearRing
><coordinates
>0,0 0,1 1,1 1,0 0,0</coordinates
></LinearRing
></outerBoundaryIs
></Polygon>

--3d linestring
SELECT ST_AsKML('SRID=4326;LINESTRING(1 2 3, 4 5 6)');
 <LineString
><coordinates
>1,2,3 4,5,6</coordinates
></LineString>
```

## Ver también

[ST\\_AsSVG](#), [ST\\_AsGML](#)

### 7.9.3.7 ST\_AsLatLonText

**ST\_AsLatLonText** — Return the Degrees, Minutes, Seconds representation of the given point.

## Synopsis

text **ST\_AsLatLonText**(geometry pt, text format=’');

## Descripción

Returns the Degrees, Minutes, Seconds representation of the point.



### Note

It is assumed the point is in a lat/lon projection. The X (lon) and Y (lat) coordinates are normalized in the output to the "normal" range (-180 to +180 for lon, -90 to +90 for lat).

The text parameter is a format string containing the format for the resulting text, similar to a date format string. Valid tokens are "D" for degrees, "M" for minutes, "S" for seconds, and "C" for cardinal direction (NSEW). DMS tokens may be repeated to indicate desired width and precision ("SSS.SSSS" means "1.0023").

"M", "S", and "C" are optional. If "C" is omitted, degrees are shown with a "-" sign if south or west. If "S" is omitted, minutes will be shown as decimal with as many digits of precision as you specify. If "M" is also omitted, degrees are shown as decimal with as many digits precision as you specify.

If the format string is omitted (or zero-length) a default format will be used.

Disponibilidad: 2.0

## Ejemplos

Default format.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)'));
 st_aslatlon

2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

Providing a format (same as the default).

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D\textdegree{}M'S.SSS"C'));
 st_aslatlon

2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

Characters other than D, M, S, C and . are just passed through.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D degrees, M minutes, S seconds to the C'));
 st_aslatlon

2 degrees, 19 minutes, 30 seconds to the S 3 degrees, 14 minutes, 3 seconds to the W
```

Signed degrees instead of cardinal directions.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D\textdegree{}M'S.SSS"));
 st_aslatlon

-2\textdegree{}19'29.928" -3\textdegree{}14'3.243"
```

Decimal degrees.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
 st_aslatlon

2.3250 degrees S 3.2342 degrees W
```

Excessively large values are normalized.

```
SELECT (ST_AsLatLonText('POINT (-302.2342342 -792.32498)'));
 st_aslatlon

72\textdegree{}19'29.928"S 57\textdegree{}45'56.757"E
```

### 7.9.3.8 ST\_AsMARC21

**ST\_AsMARC21** — Returns geometry as a MARC21/XML record with a geographic datafield (034).

#### Synopsis

text **ST\_AsMARC21** ( geometry geom , text format='hddmmss' );

#### Descripción

This function returns a MARC21/XML record with **Coded Cartographic Mathematical Data** representing the bounding box of a given geometry. The `format` parameter allows to encode the coordinates in subfields \$d,\$e,\$f and \$g in all formats supported by the MARC21/XML standard. Valid formats are:

- cardinal direction, degrees, minutes and seconds (default): `hdddmss`
- decimal degrees with cardinal direction: `hddd.dddddd`
- decimal degrees without cardinal direction: `ddd.dddddd`
- decimal minutes with cardinal direction: `hdddm.mmm`
- decimal minutes without cardinal direction: `dddm.mmm`
- decimal seconds with cardinal direction: `hdddmss.sss`

The decimal sign may be also a comma, e.g. `hdddm,mmm`.

The precision of decimal formats can be limited by the number of characters after the decimal sign, e.g. `hdddm.mm` for decimal minutes with a precision of two decimals.

This function ignores the Z and M dimensions.

LOC MARC21/XML versions supported:

- [MARC21/XML 1.1](#)

Availability: 3.3.0



#### Note

This function does not support non lon/lat geometries, as they are not supported by the MARC21/XML standard (Coded Cartographic Mathematical Data).



#### Note

The MARC21/XML Standard does not provide any means to annotate the spatial reference system for Coded Cartographic Mathematical Data, which means that this information will be lost after conversion to MARC21/XML.

## Ejemplos

Converting a POINT to MARC21/XML formatted as `hdddmss` (default)

```
SELECT ST_AsMARC21('SRID=4326;POINT(-4.504289 54.253312)::geometry');

 st_asmarc21

<record xmlns="http://www.loc.gov/MARC21/slim">
 <datafield tag="034" ind1="1" ind2=" ">
 <subfield code="a"
>a</subfield>
 <subfield code="d"
>W0043015</subfield>
 <subfield code="e"
>W0043015</subfield>
 <subfield code="f"
>N0541512</subfield>
 <subfield code="g"
>N0541512</subfield>
 </datafield>
</record>
```

### Converting a POLYGON to MARC21/XML formatted in decimal degrees

```

SELECT ST_AsMARC21('SRID=4326;POLYGON((-4.5792388916015625 ↵
54.18172660239091,-4.56756591796875 ↵
54.196993557130355,-4.546623229980469 ↵
54.18313300502024,-4.5792388916015625 54.18172660239091))'::geometry,' ↵
hddd.dddd');

<record xmlns="http://www.loc.gov/MARC21/slim">
 <datafield tag="034" ind1="1" ind2=" ">
 <subfield code="a"
>a</subfield>
 <subfield code="d"
>W004.5792</subfield>
 <subfield code="e"
>W004.5466</subfield>
 <subfield code="f"
>N054.1970</subfield>
 <subfield code="g"
>N054.1817</subfield>
 </datafield>
</record>

```

Converting a GEOMETRYCOLLECTION to MARC21/XML formatted in decimal minutes. The geometries order in the MARC21/XML output correspond to their order in the collection.

```

SELECT ST_AsMARC21('SRID=4326;GEOMETRYCOLLECTION(POLYGON((13.1 ↵
52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1 ↵
52.38333333333333,13.1 52.65)),POINT(-4.5 54.25))'::geometry,'hdddmm. ↵
mmmm');

 st_asmarc21

<record xmlns="http://www.loc.gov/MARC21/slim">
 <datafield tag="034" ind1="1" ind2=" ">
 <subfield code="a"
>a</subfield>
 <subfield code="d"
>E01307.0000</subfield>
 <subfield code="e"
>E01331.0000</subfield>
 <subfield code="f"
>N05240.0000</subfield>
 <subfield code="g"
>N05224.0000</subfield>
 </datafield>
 <datafield tag="034" ind1="1" ind2=" ">
 <subfield code="a"
>a</subfield>
 <subfield code="d"
>W00430.0000</subfield>
 <subfield code="e"
>W00430.0000</subfield>
 <subfield code="f"
>N05415.0000</subfield>
 <subfield code="g"
>N05415.0000</subfield>
 </datafield>

```

```
</record>
```

## Ver también

[ST\\_GeomFromMARC21](#)

### 7.9.3.9 ST\_AsMVTGeom

ST\_AsMVTGeom — Transforms a geometry into the coordinate space of a MVT tile.

## Synopsis

geometry **ST\_AsMVTGeom**(geometry geom, box2d bounds, integer extent=4096, integer buffer=256, boolean clip\_geom=true);

## Descripción

Transforms a geometry into the coordinate space of a MVT ([Mapbox Vector Tile](#)) tile, clipping it to the tile bounds if required. The geometry must be in the coordinate system of the target map (using [ST\\_Transform](#) if needed). Commonly this is [Web Mercator](#) (SRID:3857).

The function attempts to preserve geometry validity, and corrects it if needed. This may cause the result geometry to collapse to a lower dimension.

The rectangular bounds of the tile in the target map coordinate space must be provided, so the geometry can be transformed, and clipped if required. The bounds can be generated using [ST\\_MakeEnvelope](#).

This function is used to convert geometry into the tile coordinate space required by [ST\\_AsMVT](#).

`geom` is the geometry to transform, in the coordinate system of the target map.

`bounds` is the rectangular bounds of the tile in map coordinate space, with no buffer.

`extent` is the tile extent size in tile coordinate space as defined by the [MVT specification](#). Defaults to 4096.

`buffer` is the buffer size in tile coordinate space for geometry clipping. Defaults to 256.

`clip_geom` is a boolean to control if geometries are clipped or encoded as-is. Defaults to true.

Availability: 2.4.0



## Note

From 3.0, Wagyu can be chosen at configure time to clip and validate MVT polygons. This library is faster and produces more correct results than the GEOS default, but it might drop small polygons.

## Ejemplos

```
SELECT ST_AsText(ST_AsMVTGeom(
 ST_GeomFromText('POLYGON ((0 0, 10 0, 10 5, 0 -5, 0 0))'),
 ST_MakeBox2D(ST_Point(0, 0), ST_Point(4096, 4096)),
 4096, 0, false));
 st_astext

MULTIPOLYGON(((5 4096,10 4091,10 4096,5 4096)),((5 4096,0 4101,0 4096,5 4096)))
```

Canonical example for a Web Mercator tile using a computed tile bounds to query and clip geometry.



```
SELECT ST_AsMVTGeom(
 ST_Transform(geom, 3857),
 ST_TileEnvelope(12, 513, 412), extent =
> 4096, buffer =
> 64) AS geom
FROM data
WHERE geom && ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096))
```

## Ver también

[ST\\_AsMVT](#), [ST\\_MakeEnvelope](#), [PostGIS\\_Wagyu\\_Version](#)

### 7.9.3.10 ST\_AsMVT

**ST\_AsMVT** — Aggregate function returning a MVT representation of a set of rows.

## Synopsis

```
bytea ST_AsMVT(anyelement set row);
bytea ST_AsMVT(anyelement row, text name);
bytea ST_AsMVT(anyelement row, text name, integer extent);
bytea ST_AsMVT(anyelement row, text name, integer extent, text geom_name);
bytea ST_AsMVT(anyelement row, text name, integer extent, text geom_name, text feature_id_name);
```

## Descripción

An aggregate function which returns a binary [Mapbox Vector Tile](#) representation of a set of rows corresponding to a tile layer. The rows must contain a geometry column which will be encoded as a feature geometry. The geometry must be in tile coordinate space and valid as per the [MVT specification](#). [ST\\_AsMVTGeom](#) can be used to transform geometry into tile coordinate space. Other row columns are encoded as feature attributes.

The [Mapbox Vector Tile](#) format can store features with varying sets of attributes. To use this capability supply a JSONB column in the row data containing Json objects one level deep. The keys and values in the JSONB values will be encoded as feature attributes.

Tiles with multiple layers can be created by concatenating multiple calls to this function using `||` or `STRING_AGG`.



### Important

Do not call with a `GEOMETRYCOLLECTION` as an element in the row. However you can use [ST\\_AsMVTGeom](#) to prepare a geometry collection for inclusion.

`row` row data with at least a geometry column.

`name` is the name of the layer. Default is the string "default".

`extent` is the tile extent in screen space as defined by the specification. Default is 4096.

`geom_name` is the name of the geometry column in the row data. Default is the first geometry column. Note that PostgreSQL by default automatically [folds unquoted identifiers to lower case](#), which means that unless the geometry column is quoted, e.g. "MyMVTGeom", this parameter must be provided as lowercase.

`feature_id_name` is the name of the Feature ID column in the row data. If NULL or negative the Feature ID is not set. The first column matching name and valid type (smallint, integer, bigint) will be used as Feature ID, and any subsequent column will be added as a property. JSON properties are not supported.

Enhanced: 3.0 - added support for Feature ID.

Enhanced: 2.5.0 - added support parallel query.

Availability: 2.4.0

## Ejemplos

```
WITH mvtgeom AS
(
 SELECT ST_AsMVTGeom(geom, ST_TileEnvelope(12, 513, 412), extent =
> 4096, buffer =
> 64) AS geom, name, description
 FROM points_of_interest
 WHERE geom && ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096))
)
SELECT ST_AsMVT(mvtgeom.*)
FROM mvtgeom;
```

## Ver también

[ST\\_AsMVTGeom](#), [ST\\_MakeEnvelope](#)

### 7.9.3.11 ST\_AsSVG

**ST\_AsSVG** — Returns SVG path data for a geometry.

## Synopsis

text **ST\_AsSVG**(geometry geom, integer rel=0, integer maxdecimaldigits=15);  
text **ST\_AsSVG**(geography geog, integer rel=0, integer maxdecimaldigits=15);

## Descripción

Return the geometry as Scalar Vector Graphics (SVG) path data. Use 1 as second argument to have the path data implemented in terms of relative moves, the default (or 0) uses absolute moves. Third argument may be used to reduce the maximum number of decimal digits used in output (defaults to 15). Point geometries will be rendered as cx/cy when 'rel' arg is 0, x/y when 'rel' is 1. Multipoint geometries are delimited by commas (","), GeometryCollection geometries are delimited by semicolons (";").

For working with PostGIS SVG graphics, checkout [pg\\_svg](#) library which provides plpgsql functions for working with outputs from ST\_AsSVG.

Enhanced: 3.4.0 to support all curve types

Changed: 2.0.0 to use default args and support named args



### Note

Availability: 1.2.2. Availability: 1.4.0 Changed in PostGIS 1.4.0 to include L command in absolute path to conform to <http://www.w3.org/TR/SVG/paths.html#PathDataBNF>



This method supports Circular Strings and Curves.

## Ejemplos

```
SELECT ST_AsSVG('POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry);
```

```
st_assvg
```

```

```

```
M 0 0 L 0 -1 1 -1 1 0 Z
```

### Circular string

```
SELECT ST_AsSVG(ST_GeomFromText('CIRCULARSTRING(-2 0,0 2,2 0,0 2,2 4)'));
```

```
st_assvg
```

```

```

```
M -2 0 A 2 2 0 0 1 2 0 A 2 2 0 0 1 2 -4
```

### Multi-curve

```
SELECT ST_AsSVG('MULTICURVE((5 5,3 5,3 3,0 3),
CIRCULARSTRING(0 0,2 1,2 2))'::geometry, 0, 0);
```

```
st_assvg
```

```

```

```
M 5 -5 L 3 -5 3 -3 0 -3 M 0 0 A 2 2 0 0 0 2 -2
```

### Multi-surface

```
SELECT ST_AsSVG('MULTISURFACE(
CURVEPOLYGON(CIRCULARSTRING(-2 0,-1 -1,0 0,1 -1,2 0,0 2,-2 0),
(-1 0,0 0.5,1 0,0 1,-1 0)),
((7 8,10 10,6 14,4 11,7 8)))'::geometry, 0, 2);
```

```
st_assvg
```

```

```

```
M -2 0 A 1 1 0 0 0 0 0 A 1 1 0 0 0 2 0 A 2 2 0 0 0 -2 0 Z
```

```
M -1 0 L 0 -0.5 1 0 0 -1 -1 0 Z
```

```
M 7 -8 L 10 -10 6 -14 4 -11 Z
```

## 7.9.3.12 ST\_AsTWKB

ST\_AsTWKB — Returns the geometry as TWKB, aka "Tiny Well-Known Binary"

### Synopsis

bytea **ST\_AsTWKB**(geometry geom, integer prec=0, integer prec\_z=0, integer prec\_m=0, boolean with\_sizes=false, boolean with\_boxes=false);

bytea **ST\_AsTWKB**(geometry[] geom, bigint[] ids, integer prec=0, integer prec\_z=0, integer prec\_m=0, boolean with\_sizes=false, boolean with\_boxes=false);

### Descripción

Returns the geometry in TWKB (Tiny Well-Known Binary) format. TWKB is a **compressed binary format** with a focus on minimizing the size of the output.

The decimal digits parameters control how much precision is stored in the output. By default, values are rounded to the nearest unit before encoding. If you want to transfer more precision, increase the number. For example, a value of 1 implies that the first digit to the right of the decimal point will be preserved.

The sizes and bounding boxes parameters control whether optional information about the encoded length of the object and the bounds of the object are included in the output. By default they are not. Do not turn them on unless your client software has a use for them, as they just use up space (and saving space is the point of TWKB).

The array-input form of the function is used to convert a collection of geometries and unique identifiers into a TWKB collection that preserves the identifiers. This is useful for clients that expect to unpack a collection and then access further information about the objects inside. You can create the arrays using the [array\\_agg](#) function. The other parameters operate the same as for the simple form of the function.



#### Note

The format specification is available online at <https://github.com/TWKB/Specification>, and code for building a JavaScript client can be found at <https://github.com/TWKB/twkb.js>.

Enhanced: 2.4.0 memory and speed improvements.

Disponibilidad: 2.2.0

### Ejemplos

```
SELECT ST_AsTWKB('LINESTRING(1 1,5 5)::geometry');
 st_astwkb

\x02000202020808
```

To create an aggregate TWKB object including identifiers aggregate the desired geometries and objects first, using "array\_agg()", then call the appropriate TWKB function.

```
SELECT ST_AsTWKB(array_agg(geom), array_agg(gid)) FROM mytable;
 st_astwkb

\x040402020400000202
```

### Ver también

[ST\\_GeomFromTWKB](#), [ST\\_AsBinary](#), [ST\\_AsEWKB](#), [ST\\_AsEWKT](#), [ST\\_GeomFromText](#)

### 7.9.3.13 ST\_AsX3D

**ST\_AsX3D** — Returns a Geometry in X3D xml node element format: ISO-IEC-19776-1.2-X3DEncodings-XML

#### Synopsis

text **ST\_AsX3D**(geometry g1, integer maxdecimaldigits=15, integer options=0);

#### Descripción

Returns a geometry as an X3D xml formatted node element <http://www.web3d.org/standards/number/19776-1>. If maxdecimaldigit (precision) is not specified then defaults to 15.

**Note**

There are various options for translating PostGIS geometries to X3D since X3D geometry types don't map directly to PostGIS geometry types and some newer X3D types that might be better mappings we have avoided since most rendering tools don't currently support them. These are the mappings we have settled on. Feel free to post a bug ticket if you have thoughts on the idea or ways we can allow people to denote their preferred mappings. Below is how we currently map PostGIS 2D/3D types to X3D types

The 'options' argument is a bitfield. For PostGIS 2.2+, this is used to denote whether to represent coordinates with X3D GeoCoordinates Geospatial node and also whether to flip the x/y axis. By default, ST\_AsX3D outputs in database form (long,lat or X,Y), but X3D default of lat/lon, y/x may be preferred.

- 0: X/Y in database order (e.g. long/lat = X,Y is standard database order), default value, and non-spatial coordinates (just regular old Coordinate tag).
- 1: Flip X and Y. If used in conjunction with the GeoCoordinate option switch, then output will be default "latitude\_first" and coordinates will be flipped as well.
- 2: Output coordinates in GeoSpatial GeoCoordinates. This option will throw an error if geometries are not in WGS 84 long lat (srid: 4326). This is currently the only GeoCoordinate type supported. [Refer to X3D specs specifying a spatial reference system..](#) Default output will be GeoCoordinate geoSystem=' "GD" "WE" "longitude\_first"''. If you prefer the X3D default of GeoCoordinate geoSystem=' "GD" "WE" "latitude\_first"' use (2 + 1) = 3

| PostGIS Type                       | 2D X3D Type                              | 3D X3D Type                                                      |
|------------------------------------|------------------------------------------|------------------------------------------------------------------|
| LINestring                         | not yet implemented - will be PolyLine2D | LineSet                                                          |
| MULTILINESTRING                    | not yet implemented - will be PolyLine2D | IndexedLineSet                                                   |
| MULTIPOINT                         | Polypoint2D                              | PointSet                                                         |
| POINT                              | outputs the space delimited coordinates  | outputs the space delimited coordinates                          |
| (MULTI) POLYGON, POLYHEDRALSURFACE | Invalid X3D markup                       | IndexedFaceSet (inner rings currently output as another faceset) |
| TIN                                | TriangleSet2D (Not Yet Implemented)      | IndexedTriangleSet                                               |

**Note**

2D geometry support not yet complete. Inner rings currently just drawn as separate polygons. We are working on these.

Lots of advancements happening in 3D space particularly with [X3D Integration with HTML5](#)

There is also a nice open source X3D viewer you can use to view rendered geometries. Free Wrl <http://freewrl.sourceforge.net/> binaries available for Mac, Linux, and Windows. Use the FreeWRL\_Launcher packaged to view the geometries.

Also check out [PostGIS minimalist X3D viewer](#) that utilizes this function and [x3dDom html/js open source toolkit](#).

Availability: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML

Enhanced: 2.2.0: Support for GeoCoordinates and axis (x/y, long/lat) flipping. Look at options for details.



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



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**Example: Create a fully functional X3D document - This will generate a cube that is viewable in FreeWrl and other X3D viewers.**

```
SELECT '<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d ←
-3.0.dtd">
<X3D>
 <Scene>
 <Transform>
 <Shape>
 <Appearance>
 <Material emissiveColor=''0 0 1''/>
 </Appearance>
 </Shape>
 </Transform>
 </Scene>
</X3D>
>' As x3ddoc;

x3ddoc

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d ←
-3.0.dtd">
<X3D>
 <Scene>
 <Transform>
 <Shape>
 <Appearance>
 <Material emissiveColor='0 0 1' />
 </Appearance>
 <IndexedFaceSet coordIndex='0 1 2 3 -1 4 5 6 7 -1 8 9 10 11 -1 12 13 14 15 -1 16 17 ←
18 19 -1 20 21 22 23'>
 <Coordinate point='0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0 ←
1 0 1 0 0 1 1 1 0 1 1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 ←
1 0 1 1' />
 </IndexedFaceSet>
 </Shape>
 </Transform>
 </Scene>
</X3D>
>
```

## PostGIS buildings

Copy and paste the output of this query to [x3d scene viewer](#) and click Show

```
SELECT string_agg('<Shape
>' || ST_AsX3D(ST_Extrude(geom, 0,0, i*0.5)) ||
 '<Appearance>
 <Material diffuseColor='' || (0.01*i)::text || ' 0.8 0.2" specularColor='' || ←
 (0.05*i)::text || ' 0 0.5"/>
 </Appearance>
</Shape
>', '')
FROM ST_Subdivide(ST_Letters('PostGIS'),20) WITH ORDINALITY AS f(geom,i);
```



*Buildings formed by subdividing PostGIS and extrusion*

#### Example: An Octagon elevated 3 Units and decimal precision of 6

```
SELECT ST_AsX3D(
ST_Translate(
 ST_Force_3d(
 ST_Buffer(ST_Point(10,10),5, 'quad_segs=2')), 0,0,
 3)
,6) As x3dfrag;

x3dfrag

<IndexedFaceSet coordIndex="0 1 2 3 4 5 6 7">
 <Coordinate point="15 10 3 13.535534 6.464466 3 10 5 3 6.464466 6.464466 3 5 10 3 ←
 6.464466 13.535534 3 10 15 3 13.535534 13.535534 3 " />
</IndexedFaceSet
>
```

#### Ejemplo: TIN

```
SELECT ST_AsX3D(ST_GeomFromEWKT('TIN (((
 0 0 0,
 0 0 1,
 0 1 0,
 0 0 0
)), ((
 0 0 0,
 0 1 0,
 1 1 0,
 0 0 0
))
)')) As x3dfrag;

x3dfrag

<IndexedTriangleSet index='0 1 2 3 4 5'
><Coordinate point='0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0' /></IndexedTriangleSet
>
```

#### Example: Closed multilinestring (the boundary of a polygon with holes)

```
SELECT ST_AsX3D(
 ST_GeomFromEWKT('MULTILINESTRING((20 0 10,16 -12 10,0 -16 10,-12 -12 ←
 10,-20 0 10,-12 16 10,0 24 10,16 16 10,20 0 10),
 (12 0 10,8 8 10,0 12 10,-8 8 10,-8 0 10,-8 -4 10,0 -8 10,8 -4 10,12 0 10))')
)
```

```

) As x3dfrag;

 x3dfrag

<IndexedLineSet coordIndex='0 1 2 3 4 5 6 7 0 -1 8 9 10 11 12 13 14 15 8'>
 <Coordinate point='20 0 10 16 -12 10 0 -16 10 -12 -12 10 -20 0 10 -12 16 10 0 24 10 16 ←
 16 10 12 0 10 8 8 10 0 12 10 -8 8 10 -8 0 10 -8 -4 10 0 -8 10 8 -4 10 ' />
</IndexedLineSet
>

```

### 7.9.3.14 ST\_GeoHash

ST\_GeoHash — Return a GeoHash representation of the geometry.

#### Synopsis

text **ST\_GeoHash**(geometry geom, integer maxchars=full\_precision\_of\_point);

#### Descripción

Computes a **GeoHash** representation of a geometry. A GeoHash encodes a geographic Point into a text form that is sortable and searchable based on prefixing. A shorter GeoHash is a less precise representation of a point. It can be thought of as a box that contains the point.

Non-point geometry values with non-zero extent can also be mapped to GeoHash codes. The precision of the code depends on the geographic extent of the geometry.

If `maxchars` is not specified, the returned GeoHash code is for the smallest cell containing the input geometry. Points return a GeoHash with 20 characters of precision (about enough to hold the full double precision of the input). Other geometric types may return a GeoHash with less precision, depending on the extent of the geometry. Larger geometries are represented with less precision, smaller ones with more precision. The box determined by the GeoHash code always contains the input feature.

If `maxchars` is specified the returned GeoHash code has at most that many characters. It maps to a (possibly) lower precision representation of the input geometry. For non-points, the starting point of the calculation is the center of the bounding box of the geometry.

Disponibilidad: 1.4.0



#### Note

ST\_GeoHash requires input geometry to be in geographic (lon/lat) coordinates.



This method supports Circular Strings and Curves.

#### Ejemplos

```

SELECT ST_GeoHash(ST_Point(-126,48));

 st_geohash

c0w3hf1s70w3hf1s70w3

SELECT ST_GeoHash(ST_Point(-126,48), 5);

 st_geohash

```



```

c0w3h

-- This line contains the point, so the GeoHash is a prefix of the point code
SELECT ST_GeoHash('LINESTRING(-126 48, -126.1 48.1)::geometry);

st_geohash

c0w3

```

**Ver también**

[ST\\_GeomFromGeoHash](#), [ST\\_PointFromGeoHash](#), [ST\\_Box2dFromGeoHash](#)

## 7.10 Operadores

### 7.10.1 Bounding Box Operators

#### 7.10.1.1 &&

**&&** — Returns TRUE if A's 2D bounding box intersects B's 2D bounding box.

#### Synopsis

```

boolean &&(geometry A , geometry B);
boolean &&(geography A , geography B);

```

#### Descripción

The **&&** operator returns TRUE if the 2D bounding box of geometry A intersects the 2D bounding box of geometry B.



#### Note

This operand will make use of any indexes that may be available on the geometries.

Mejorado: 2.0.0 soporte para superficies poliédricas fue introducida.

Availability: 1.5.0 support for geography was introduced.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

#### Ejemplos

```

SELECT tbl1.column1, tbl2.column1, tbl1.column2 && tbl2.column2 AS overlaps
FROM (VALUES
 (1, 'LINESTRING(0 0, 3 3)::geometry)',
 (2, 'LINESTRING(0 1, 0 5)::geometry)) AS tbl1,
(VALUES
 (3, 'LINESTRING(1 2, 4 6)::geometry)) AS tbl2;

```

```

column1 | column1 | overlaps
-----+-----+-----
 1 | 3 | t
 2 | 3 | f
(2 rows)

```

### Ver también

[ST\\_Intersects](#), [ST\\_Extent](#), [l&>](#), [&>](#), [&<l](#), [&<](#), [~](#), [@](#)

#### 7.10.1.2 &&(geometry,box2df)

**&&(geometry,box2df)** — Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

### Synopsis

boolean **&&**( geometry A , box2df B );

### Descripción

The **&&** operator returns TRUE if the cached 2D bounding box of geometry A intersects the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



#### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

### Ejemplos

```

SELECT ST_Point(1,1) && ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS overlaps;

 overlaps

 t
(1 row)

```

### Ver también

[&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

### 7.10.1.3 &&(box2df,geometry)

**&&(box2df,geometry)** — Returns `TRUE` if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.

#### Synopsis

boolean **&&**( box2df A , geometry B );

#### Descripción

The **&&** operator returns `TRUE` if the 2D bounding box A intersects the cached 2D bounding box of geometry B, using float precision. This means that if A is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



#### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

#### Ejemplos

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_Point(1,1) AS overlaps;

overlaps

t
(1 row)
```

#### Ver también

[&&\(geometry,box2df\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

### 7.10.1.4 &&(box2df,box2df)

**&&(box2df,box2df)** — Returns `TRUE` if two 2D float precision bounding boxes (BOX2DF) intersect each other.

#### Synopsis

boolean **&&**( box2df A , box2df B );

## Descripción

The `&&` operator returns `TRUE` if two 2D bounding boxes A and B intersect each other, using float precision. This means that if A (or B) is a (double precision) `box2d`, it will be internally converted to a float precision 2D bounding box (`BOX2DF`)



### Note

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

## Ejemplos

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_MakeBox2D(ST_Point(1,1), ST_Point(3,3)) AS overlaps;

overlaps

t
(1 row)
```

## Ver también

`&&(geometry,box2df)`, `&&(box2df,geometry)`, `~(geometry,box2df)`, `~(box2df,geometry)`, `~(box2df,box2df)`, `@(geometry,box2df)`, `@(box2df,geometry)`, `@(box2df,box2df)`

### 7.10.1.5 &&&

`&&&` — Returns `TRUE` if A's n-D bounding box intersects B's n-D bounding box.

## Synopsis

boolean `&&&( geometry A , geometry B );`

## Descripción

The `&&&` operator returns `TRUE` if the n-D bounding box of geometry A intersects the n-D bounding box of geometry B.



### Note

This operand will make use of any indexes that may be available on the geometries.

Disponibilidad: 2.0.0



This method supports Circular Strings and Curves.

- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✔ This function supports 3d and will not drop the z-index.

**Examples: 3D LineStrings**

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &&& tbl2.column2 AS overlaps_3d,
 tbl1.column2 && tbl2.column2 AS overlaps_2d
FROM (VALUES
 (1, 'LINESTRING Z(0 0 1, 3 3 2)::geometry),
 (2, 'LINESTRING Z(1 2 0, 0 5 -1)::geometry)) AS tbl1,
(VALUES
 (3, 'LINESTRING Z(1 2 1, 4 6 1)::geometry)) AS tbl2;
```

| column1 | column1 | overlaps_3d | overlaps_2d |
|---------|---------|-------------|-------------|
| 1       | 3       | t           | t           |
| 2       | 3       | f           | t           |

**Examples: 3M LineStrings**

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &&& tbl2.column2 AS overlaps_3zm,
 tbl1.column2 && tbl2.column2 AS overlaps_2d
FROM (VALUES
 (1, 'LINESTRING M(0 0 1, 3 3 2)::geometry),
 (2, 'LINESTRING M(1 2 0, 0 5 -1)::geometry)) AS tbl1,
(VALUES
 (3, 'LINESTRING M(1 2 1, 4 6 1)::geometry)) AS tbl2;
```

| column1 | column1 | overlaps_3zm | overlaps_2d |
|---------|---------|--------------|-------------|
| 1       | 3       | t            | t           |
| 2       | 3       | f            | t           |

**Ver también**

**&&**

**7.10.1.6 &&&(geometry,gidx)**

**&&&(geometry,gidx)** — Returns TRUE if a geometry’s (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).

**Synopsis**

boolean **&&&**( geometry A , gidx B );

## Descripción

The `&&&` operator returns `TRUE` if the cached n-D bounding box of geometry A intersects the n-D bounding box B, using float precision. This means that if B is a (double precision) `box3d`, it will be internally converted to a float precision 3D bounding box (GIDX)



### Note

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



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

## Ejemplos

```
SELECT ST_MakePoint(1,1,1) &&& ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) AS overlaps;

overlaps

t
(1 row)
```

## Ver también

[`&&&\(gidx,geometry\), &&&\(gidx,gidx\)`](#)

### 7.10.1.7 `&&&(gidx,geometry)`

`&&&(gidx,geometry)` — Returns `TRUE` if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.

## Synopsis

boolean `&&&( gidx A , geometry B );`

## Descripción

The `&&&` operator returns `TRUE` if the n-D bounding box A intersects the cached n-D bounding box of geometry B, using float precision. This means that if A is a (double precision) `box3d`, it will be internally converted to a float precision 3D bounding box (GIDX)



### Note

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



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

## Ejemplos

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_MakePoint(1,1,1) AS overlaps;
overlaps

t
(1 row)
```

## Ver también

[&&&\(geometry,gidx\), &&&\(gidx,gidx\)](#)

### 7.10.1.8 &&&(gidx,gidx)

[&&&\(gidx,gidx\)](#) — Returns `TRUE` if two n-D float precision bounding boxes (GIDX) intersect each other.

## Synopsis

boolean [&&&](#)( gidx A , gidx B );

## Descripción

The [&&&](#) operator returns `TRUE` if two n-D bounding boxes A and B intersect each other, using float precision. This means that if A (or B) is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)



### Note

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



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

Ejemplos

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_3DMakeBox(ST_MakePoint(1,1,1), ST_MakePoint(3,3,3)) AS overlaps;

overlaps

t
(1 row)
```

Ver también

[&&&\(geometry,gidx\), &&&\(gidx,geometry\)](#)

7.10.1.9 &<

&< — Returns TRUE if A's bounding box overlaps or is to the left of B's.

Synopsis

boolean **&<**( geometry A , geometry B );

Descripción

The &< operator returns TRUE if the bounding box of geometry A overlaps or is to the left of the bounding box of geometry B, or more accurately, overlaps or is NOT to the right of the bounding box of geometry B.



**Note**  
This operand will make use of any indexes that may be available on the geometries.

Ejemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &< tbl2.column2 AS overleft
FROM
 (VALUES
 (1, 'LINESTRING(1 2, 4 6)::geometry)) AS tbl1,
 (VALUES
 (2, 'LINESTRING(0 0, 3 3)::geometry),
 (3, 'LINESTRING(0 1, 0 5)::geometry),
 (4, 'LINESTRING(6 0, 6 1)::geometry)) AS tbl2;

column1 | column1 | overleft
-----+-----+-----
 1 | 2 | f
 1 | 3 | f
 1 | 4 | t
(3 rows)
```

Ver también

[&&, |&>, &>, &<|](#)



7.10.1.10 &<|

&<| — Returns TRUE if A’s bounding box overlaps or is below B’s.

Synopsis

boolean &<|( geometry A , geometry B );

Descripción

The &<| operator returns TRUE if the bounding box of geometry A overlaps or is below of the bounding box of geometry B, or more accurately, overlaps or is NOT above the bounding box of geometry B.

-  This method supports Circular Strings and Curves.
-  This function supports Polyhedral surfaces.



Note

This operand will make use of any indexes that may be available on the geometries.

Ejemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &<| tbl2.column2 AS overbelow
FROM
 (VALUES
 (1, 'LINESTRING(6 0, 6 4)::geometry)) AS tbl1,
 (VALUES
 (2, 'LINESTRING(0 0, 3 3)::geometry),
 (3, 'LINESTRING(0 1, 0 5)::geometry),
 (4, 'LINESTRING(1 2, 4 6)::geometry)) AS tbl2;
```

| column1 | column1 | overbelow |
|---------|---------|-----------|
| 1       | 2       | f         |
| 1       | 3       | t         |
| 1       | 4       | t         |

(3 rows)

Ver también

[&&](#), [|&>](#), [&>](#), [&<](#)

7.10.1.11 &>

&> — Returns TRUE if A’ bounding box overlaps or is to the right of B’s.

Synopsis

boolean &>( geometry A , geometry B );

Descripción

The `&>` operator returns `TRUE` if the bounding box of geometry A overlaps or is to the right of the bounding box of geometry B, or more accurately, overlaps or is NOT to the left of the bounding box of geometry B.



**Note**  
This operand will make use of any indexes that may be available on the geometries.

Ejemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &> tbl2.column2 AS overright
FROM
 (VALUES
 (1, 'LINESTRING(1 2, 4 6)::geometry) AS tbl1,
 (VALUES
 (2, 'LINESTRING(0 0, 3 3)::geometry),
 (3, 'LINESTRING(0 1, 0 5)::geometry),
 (4, 'LINESTRING(6 0, 6 1)::geometry) AS tbl2;

column1 | column1 | overright
-----+-----+-----
 1 | 2 | t
 1 | 3 | t
 1 | 4 | f
(3 rows)
```

Ver también

`&&`, `|&>`, `&<|`, `&<`

7.10.1.12 <<

`<<` — Returns `TRUE` if A's bounding box is strictly to the left of B's.

Synopsis

boolean `<<`( geometry A , geometry B );

Descripción

The `<<` operator returns `TRUE` if the bounding box of geometry A is strictly to the left of the bounding box of geometry B.



**Note**  
This operand will make use of any indexes that may be available on the geometries.

Ejemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 << tbl2.column2 AS left
FROM
 (VALUES
 (1, 'LINESTRING (1 2, 1 5)::geometry)) AS tbl1,
 (VALUES
 (2, 'LINESTRING (0 0, 4 3)::geometry),
 (3, 'LINESTRING (6 0, 6 5)::geometry),
 (4, 'LINESTRING (2 2, 5 6)::geometry)) AS tbl2;

column1 | column1 | left
-----+-----+-----
 1 | 2 | f
 1 | 3 | t
 1 | 4 | t
(3 rows)
```

Ver también

>>, |>>, <<|

7.10.1.13 <<|

<<| — Returns TRUE if A’s bounding box is strictly below B’s.

Synopsis

boolean <<|( geometry A , geometry B );

Descripción

The <<| operator returns TRUE if the bounding box of geometry A is strictly below the bounding box of geometry B.



Note

This operand will make use of any indexes that may be available on the geometries.

Ejemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 <<| tbl2.column2 AS below
FROM
 (VALUES
 (1, 'LINESTRING (0 0, 4 3)::geometry)) AS tbl1,
 (VALUES
 (2, 'LINESTRING (1 4, 1 7)::geometry),
 (3, 'LINESTRING (6 1, 6 5)::geometry),
 (4, 'LINESTRING (2 3, 5 6)::geometry)) AS tbl2;

column1 | column1 | below
-----+-----+-----
 1 | 2 | t
 1 | 3 | f
 1 | 4 | f
(3 rows)
```

**Ver también**

&lt;&lt;, &gt;&gt;, |&gt;&gt;

**7.10.1.14 =**

= — Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.

**Synopsis**

```
boolean =(geometry A , geometry B);
boolean =(geography A , geography B);
```

**Descripción**

The = operator returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B. PostgreSQL uses the =, <, and > operators defined for geometries to perform internal orderings and comparison of geometries (ie. in a GROUP BY or ORDER BY clause).

**Note**

Only geometry/geography that are exactly equal in all respects, with the same coordinates, in the same order, are considered equal by this operator. For "spatial equality", that ignores things like coordinate order, and can detect features that cover the same spatial area with different representations, use [ST\\_OrderingEquals](#) or [ST\\_Equals](#)

**Caution**

This operand will NOT make use of any indexes that may be available on the geometries. For an index assisted exact equality test, combine = with &&.

Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use ~= instead.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

**Ejemplos**

```
SELECT 'LINESTRING(0 0, 0 1, 1 0)::geometry = 'LINESTRING(1 1, 0 0)::geometry;
?column?

f
(1 row)

SELECT ST_AsText(column1)
FROM (VALUES
 ('LINESTRING(0 0, 1 1)::geometry',
 ('LINESTRING(1 1, 0 0)::geometry') AS foo;
 st_astext

LINESTRING(0 0,1 1)
LINESTRING(1 1,0 0)
```

```

(2 rows)

-- Note: the GROUP BY uses the "=" to compare for geometry equivalency.
SELECT ST_AsText(column1)
FROM (VALUES
 ('LINESTRING(0 0, 1 1)::geometry)',
 ('LINESTRING(1 1, 0 0)::geometry')) AS foo
GROUP BY column1;
 st_astext

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

-- In versions prior to 2.0, this used to return true --
SELECT ST_GeomFromText('POINT(1707296.37 4820536.77)') =
 ST_GeomFromText('POINT(1707296.27 4820536.87)') As pt_intersect;

--pt_intersect --
f

```

### Ver también

[ST\\_Equals](#), [ST\\_OrderingEquals](#), [~=](#)

#### 7.10.1.15 >>

>> — Returns TRUE if A's bounding box is strictly to the right of B's.

### Synopsis

boolean >>( geometry A , geometry B );

### Descripción

The >> operator returns TRUE if the bounding box of geometry A is strictly to the right of the bounding box of geometry B.



#### Note

This operand will make use of any indexes that may be available on the geometries.

### Ejemplos

```

SELECT tbl1.column1, tbl2.column1, tbl1.column2 >> tbl2.column2 AS right
FROM
 (VALUES
 (1, 'LINESTRING (2 3, 5 6)::geometry)) AS tbl1,
 (VALUES
 (2, 'LINESTRING (1 4, 1 7)::geometry),
 (3, 'LINESTRING (6 1, 6 5)::geometry),
 (4, 'LINESTRING (0 0, 4 3)::geometry)) AS tbl2;

column1 | column1 | right
-----+-----+-----

```

|          |  |   |  |   |
|----------|--|---|--|---|
| 1        |  | 2 |  | t |
| 1        |  | 3 |  | f |
| 1        |  | 4 |  | f |
| (3 rows) |  |   |  |   |

Ver también

<<, >>, <<|

7.10.1.16 @

@ — Returns TRUE if A’s bounding box is contained by B’s.

Synopsis

boolean @( geometry A , geometry B );

Descripción

The @ operator returns TRUE if the bounding box of geometry A is completely contained by the bounding box of geometry B.



Note

This operand will make use of any indexes that may be available on the geometries.

Ejemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 @ tbl2.column2 AS contained
FROM
 (VALUES
 (1, 'LINESTRING (1 1, 3 3)::geometry)) AS tbl1,
 (VALUES
 (2, 'LINESTRING (0 0, 4 4)::geometry),
 (3, 'LINESTRING (2 2, 4 4)::geometry),
 (4, 'LINESTRING (1 1, 3 3)::geometry)) AS tbl2;
```

| column1  |  | column1 |  | contained |
|----------|--|---------|--|-----------|
| 1        |  | 2       |  | t         |
| 1        |  | 3       |  | f         |
| 1        |  | 4       |  | t         |
| (3 rows) |  |         |  |           |

Ver también

~, &&

7.10.1.17 @(geometry,box2df)

@(geometry,box2df) — Returns TRUE if a geometry’s 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).

## Synopsis

boolean @( geometry A , box2df B );

## Descripción

The @ operator returns TRUE if the A geometry's 2D bounding box is contained the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

## Ejemplos

```
SELECT ST_Buffer(ST_GeomFromText('POINT(2 2)'), 1) @ ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) AS is_contained;
```

```
is_contained

t
(1 row)
```

## Ver también

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@ \(box2df,geometry\)](#), [@ \(box2df,box2df\)](#)

### 7.10.1.18 @ (box2df,geometry)

@(box2df,geometry) — Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.

## Synopsis

boolean @( box2df A , geometry B );

## Descripción

The @ operator returns TRUE if the 2D bounding box A is contained into the B geometry's 2D bounding box, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

## Ejemplos

```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_Buffer(ST_GeomFromText('POINT(1 1)') ←
, 10) AS is_contained;
```

```
is_contained

t
(1 row)
```

## Ver también

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,box2df\)](#)

### 7.10.1.19 @(box2df,box2df)

@(box2df,box2df) — Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.

## Synopsis

boolean @( box2df A , box2df B );

## Descripción

The @ operator returns TRUE if the 2D bounding box A is contained into the 2D bounding box B, using float precision. This means that if A (or B) is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

## Ejemplos

```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_MakeBox2D(ST_Point(0,0), ST_Point ←
(5,5)) AS is_contained;
```

```
is_contained

t
(1 row)
```



Ver también

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@ \(geometry,box2df\)](#), [@ \(box2df,geometry\)](#)

7.10.1.20 |&>

|&> — Returns TRUE if A’s bounding box overlaps or is above B’s.

Synopsis

boolean |&>( geometry A , geometry B );

Descripción

The |&> operator returns TRUE if the bounding box of geometry A overlaps or is above the bounding box of geometry B, or more accurately, overlaps or is NOT below the bounding box of geometry B.



Note

This operand will make use of any indexes that may be available on the geometries.

Ejemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |&> tbl2.column2 AS overabove
FROM
 (VALUES
 (1, 'LINESTRING(6 0, 6 4)::geometry)) AS tbl1,
 (VALUES
 (2, 'LINESTRING(0 0, 3 3)::geometry),
 (3, 'LINESTRING(0 1, 0 5)::geometry),
 (4, 'LINESTRING(1 2, 4 6)::geometry)) AS tbl2;
```

| column1 | column1 | overabove |
|---------|---------|-----------|
| 1       | 2       | t         |
| 1       | 3       | f         |
| 1       | 4       | f         |

(3 rows)

Ver también

[&&](#), [&>](#), [&<|](#), [&<](#)

7.10.1.21 |>>

|>> — Returns TRUE if A’s bounding box is strictly above B’s.

Synopsis

boolean |>>( geometry A , geometry B );

Descripción

The `|>>` operator returns `TRUE` if the bounding box of geometry A is strictly above the bounding box of geometry B.



**Note**  
This operand will make use of any indexes that may be available on the geometries.

Ejemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |>> tbl2.column2 AS above
FROM
 (VALUES
 (1, 'LINESTRING (1 4, 1 7)::geometry') AS tbl1,
 (VALUES
 (2, 'LINESTRING (0 0, 4 2)::geometry),
 (3, 'LINESTRING (6 1, 6 5)::geometry),
 (4, 'LINESTRING (2 3, 5 6)::geometry')) AS tbl2;

column1 | column1 | above
-----+-----+-----
 1 | 2 | t
 1 | 3 | f
 1 | 4 | f
(3 rows)
```

Ver también

`<<`, `>>`, `<<|`

7.10.1.22 ~

`~` — Returns `TRUE` if A's bounding box contains B's.

Synopsis

boolean `~( geometry A , geometry B );`

Descripción

The `~` operator returns `TRUE` if the bounding box of geometry A completely contains the bounding box of geometry B.



**Note**  
This operand will make use of any indexes that may be available on the geometries.

Ejemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 ~ tbl2.column2 AS contains
FROM
 (VALUES
 (1, 'LINESTRING (0 0, 3 3)::geometry') AS tbl1,
 (VALUES
 (2, 'LINESTRING (0 0, 4 4)::geometry'),
 (3, 'LINESTRING (1 1, 2 2)::geometry'),
 (4, 'LINESTRING (0 0, 3 3)::geometry') AS tbl2;

column1 | column1 | contains
-----+-----+-----
 1 | 2 | f
 1 | 3 | t
 1 | 4 | t
(3 rows)
```

Ver también

@, &&

7.10.1.23 ~(geometry,box2df)

~(geometry,box2df) — Returns TRUE if a geometry’s 2D bonding box contains a 2D float precision bounding box (GIDX).

Synopsis

boolean ~( geometry A , box2df B );

Descripción

The ~ operator returns TRUE if the 2D bounding box of a geometry A contains the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.

Ejemplos

```
SELECT ST_Buffer(ST_GeomFromText('POINT(1 1)'), 10) ~ ST_MakeBox2D(ST_Point(0,0), ST_Point(
 (2,2)) AS contains;

contains

t
(1 row)
```

**Ver también**

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

**7.10.1.24 ~(box2df,geometry)**

`~(box2df,geometry)` — Returns `TRUE` if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.

**Synopsis**

boolean `~( box2df A , geometry B );`

**Descripción**

The `~` operator returns `TRUE` if the 2D bounding box A contains the B geometry's bounding box, using float precision. This means that if A is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

**Note**

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

**Ejemplos**

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_Buffer(ST_GeomFromText('POINT(2 2)')
, 1) AS contains;

contains

t
(1 row)
```

**Ver también**

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

**7.10.1.25 ~(box2df,box2df)**

`~(box2df,box2df)` — Returns `TRUE` if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).

**Synopsis**

boolean `~( box2df A , box2df B );`

## Descripción

The `~` operator returns `TRUE` if the 2D bounding box A contains the 2D bounding box B, using float precision. This means that if A is a (double precision) `box2d`, it will be internally converted to a float precision 2D bounding box (`BOX2DF`)



### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

## Ejemplos

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) AS contains;

contains

t
(1 row)
```

## Ver también

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

### 7.10.1.26 ~=

`~=` — Returns `TRUE` if A's bounding box is the same as B's.

## Synopsis

boolean `~=( geometry A , geometry B );`

## Descripción

The `~=` operator returns `TRUE` if the bounding box of geometry/geography A is the same as the bounding box of geometry/geography B.



### Note

This operand will make use of any indexes that may be available on the geometries.

Availability: 1.5.0 changed behavior



This function supports Polyhedral surfaces.

**Warning**

This operator has changed behavior in PostGIS 1.5 from testing for actual geometric equality to only checking for bounding box equality. To complicate things it also depends on if you have done a hard or soft upgrade which behavior your database has. To find out which behavior your database has you can run the query below. To check for true equality use [ST\\_OrderingEquals](#) or [ST\\_Equals](#).

**Ejemplos**

```
select 'LINESTRING(0 0, 1 1)::geometry' ~= 'LINESTRING(0 1, 1 0)::geometry' as equality;
equality |
-----+
t |
```

**Ver también**

[ST\\_Equals](#), [ST\\_OrderingEquals](#), [=](#)

**7.10.2 Operadores****7.10.2.1 <->**

<-> — Returns the 2D distance between A and B.

**Synopsis**

```
double precision <->(geometry A , geometry B);
double precision <->(geography A , geography B);
```

**Descripción**

The <-> operator returns the 2D distance between two geometries. Used in the "ORDER BY" clause provides index-assisted nearest-neighbor result sets. For PostgreSQL below 9.5 only gives centroid distance of bounding boxes and for PostgreSQL 9.5+, does true KNN distance search giving true distance between geometries, and distance sphere for geographies.

**Note**

This operand will make use of 2D GiST indexes that may be available on the geometries. It is different from other operators that use spatial indexes in that the spatial index is only used when the operator is in the ORDER BY clause.

**Note**

Index only kicks in if one of the geometries is a constant (not in a subquery/cte). e.g. 'SRID=3005;POINT(1011102 450541)::geometry' instead of a.geom

Refer to [PostGIS workshop: Nearest-Neighbor Searching](#) for a detailed example.

Enhanced: 2.2.0 -- True KNN ("K nearest neighbor") behavior for geometry and geography for PostgreSQL 9.5+. Note for geography KNN is based on sphere rather than spheroid. For PostgreSQL 9.4 and below, geography support is new but only supports centroid box.

Changed: 2.2.0 -- For PostgreSQL 9.5 users, old Hybrid syntax may be slower, so you'll want to get rid of that hack if you are running your code only on PostGIS 2.2+ 9.5+. See examples below.

Availability: 2.0.0 -- Weak KNN provides nearest neighbors based on geometry centroid distances instead of true distances. Exact results for points, inexact for all other types. Available for PostgreSQL 9.1+

## Ejemplos

```
SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)::geometry) as d,edabbr, vaabbr
FROM va2005
ORDER BY d limit 10;
```

| d                | edabbr | vaabbr |
|------------------|--------|--------|
| 0                | ALQ    | 128    |
| 5541.57712511724 | ALQ    | 129A   |
| 5579.67450712005 | ALQ    | 001    |
| 6083.4207708641  | ALQ    | 131    |
| 7691.2205404848  | ALQ    | 003    |
| 7900.75451037313 | ALQ    | 122    |
| 8694.20710669982 | ALQ    | 129B   |
| 9564.24289057111 | ALQ    | 130    |
| 12089.665931705  | ALQ    | 127    |
| 18472.5531479404 | ALQ    | 002    |

(10 rows)

Then the KNN raw answer:

```
SELECT st_distance(geom, 'SRID=3005;POINT(1011102 450541)::geometry) as d,edabbr, vaabbr
FROM va2005
ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)::geometry' limit 10;
```

| d                | edabbr | vaabbr |
|------------------|--------|--------|
| 0                | ALQ    | 128    |
| 5541.57712511724 | ALQ    | 129A   |
| 5579.67450712005 | ALQ    | 001    |
| 6083.4207708641  | ALQ    | 131    |
| 7691.2205404848  | ALQ    | 003    |
| 7900.75451037313 | ALQ    | 122    |
| 8694.20710669982 | ALQ    | 129B   |
| 9564.24289057111 | ALQ    | 130    |
| 12089.665931705  | ALQ    | 127    |
| 18472.5531479404 | ALQ    | 002    |

(10 rows)

If you run "EXPLAIN ANALYZE" on the two queries you would see a performance improvement for the second.

For users running with PostgreSQL < 9.5, use a hybrid query to find the true nearest neighbors. First a CTE query using the index-assisted KNN, then an exact query to get correct ordering:

```
WITH index_query AS (
 SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)::geometry) as d,edabbr, vaabbr
 FROM va2005
 ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)::geometry' LIMIT 100)
SELECT *
 FROM index_query
 ORDER BY d limit 10;
```

| d                | edabbr | vaabbr |
|------------------|--------|--------|
| 0                | ALQ    | 128    |
| 5541.57712511724 | ALQ    | 129A   |
| 5579.67450712005 | ALQ    | 001    |
| 6083.4207708641  | ALQ    | 131    |
| 7691.2205404848  | ALQ    | 003    |
| 7900.75451037313 | ALQ    | 122    |
| 8694.20710669982 | ALQ    | 129B   |

```
9564.24289057111 | ALQ | 130
12089.665931705 | ALQ | 127
18472.5531479404 | ALQ | 002
(10 rows)
```

Ver también

[ST\\_DWithin](#), [ST\\_Distance](#), [<#>](#)

7.10.2.2 `|=`

`|=` — Returns the distance between A and B trajectories at their closest point of approach.

Synopsis

double precision `|=`( geometry A , geometry B );

Descripción

The `|=` operator returns the 3D distance between two trajectories (See [ST\\_IsValidTrajectory](#)). This is the same as [ST\\_DistanceCPA](#) but as an operator it can be used for doing nearest neighbor searches using an N-dimensional index (requires PostgreSQL 9.5.0 or higher).



Note

This operand will make use of ND GiST indexes that may be available on the geometries. It is different from other operators that use spatial indexes in that the spatial index is only used when the operator is in the ORDER BY clause.



Note

Index only kicks in if one of the geometries is a constant (not in a subquery/cte). e.g. `'SRID=3005;LINESTRINGM(0 0 0,0 0 1)::geometry` instead of `a.geom`

Availability: 2.2.0. Index-supported only available for PostgreSQL 9.5+

Ejemplos

```
-- Save a literal query trajectory in a psql variable...
\set qt 'ST_AddMeasure(ST_MakeLine(ST_MakePointM(-350,300,0),ST_MakePointM(-410,490,0)) ←
,10,20) '
-- Run the query !
SELECT track_id, dist FROM (
 SELECT track_id, ST_DistanceCPA(tr,:qt) dist
 FROM trajectories
 ORDER BY tr |=| :qt
 LIMIT 5
) foo;
track_id dist
-----+-----
395 | 0.576496831518066
380 | 5.06797130410151
390 | 7.72262293958322
```



```
385 | 9.8004461358071
405 | 10.9534397988433
(5 rows)
```

Ver también

[ST\\_DistanceCPA](#), [ST\\_ClosestPointOfApproach](#), [ST\\_IsValidTrajectory](#)

7.10.2.3 <#>

<#> — Returns the 2D distance between A and B bounding boxes.

Synopsis

double precision <#>( geometry A , geometry B );

Descripción

The <#> operator returns distance between two floating point bounding boxes, possibly reading them from a spatial index (PostgreSQL 9.1+ required). Useful for doing nearest neighbor **approximate** distance ordering.



**Note**  
This operand will make use of any indexes that may be available on the geometries. It is different from other operators that use spatial indexes in that the spatial index is only used when the operator is in the ORDER BY clause.



**Note**  
Index only kicks in if one of the geometries is a constant e.g. ORDER BY (ST\_GeomFromText('POINT(1 2)') <#> geom) instead of g1.geom <#>.

Availability: 2.0.0 -- KNN only available for PostgreSQL 9.1+

Ejemplos

```
SELECT *
FROM (
SELECT b.tlid, b.mtfcc,
 b.geom <#
> ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
 745787 2948499,745740 2948468,745712 2948438,
 745690 2948384,745677 2948319)',2249) As b_dist,
 ST_Distance(b.geom, ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
 745787 2948499,745740 2948468,745712 2948438,
 745690 2948384,745677 2948319)',2249)) As act_dist
FROM bos_roads As b
ORDER BY b_dist, b.tlid
LIMIT 100) As foo
ORDER BY act_dist, tlid LIMIT 10;
```

| tlid                    | mtfcc | b_dist | act_dist |
|-------------------------|-------|--------|----------|
| -----+-----+-----+----- |       |        |          |

```

85732027 | S1400 | 0 | 0
85732029 | S1400 | 0 | 0
85732031 | S1400 | 0 | 0
85734335 | S1400 | 0 | 0
85736037 | S1400 | 0 | 0
624683742 | S1400 | 0 | 128.528874268666
85719343 | S1400 | 260.839270432962 | 260.839270432962
85741826 | S1400 | 164.759294123275 | 260.839270432962
85732032 | S1400 | 277.75 | 311.830282365264
85735592 | S1400 | 222.25 | 311.830282365264
(10 rows)

```

**Ver también****ST\_DWithin, ST\_Distance, <->****7.10.2.4 <<->****<<->** — Returns the n-D distance between the centroids of A and B bounding boxes.**Synopsis**double precision **<<->**( geometry A , geometry B );**Descripción**

The **<<->** operator returns the n-D (euclidean) distance between the centroids of the bounding boxes of two geometries. Useful for doing nearest neighbor **approximate** distance ordering.

**Note**

This operand will make use of n-D GiST indexes that may be available on the geometries. It is different from other operators that use spatial indexes in that the spatial index is only used when the operator is in the ORDER BY clause.

**Note**

Index only kicks in if one of the geometries is a constant (not in a subquery/cte). e.g. 'SRID=3005;POINT(1011102 450541)>::geometry instead of a.geom

Availability: 2.2.0 -- KNN only available for PostgreSQL 9.1+

**Ver también****<<#>>, <->****7.10.2.5 <<#>>****<<#>>** — Returns the n-D distance between A and B bounding boxes.**Synopsis**double precision **<<#>>**( geometry A , geometry B );

## Descripción

The `<<#>>` operator returns distance between two floating point bounding boxes, possibly reading them from a spatial index (PostgreSQL 9.1+ required). Useful for doing nearest neighbor **approximate** distance ordering.



### Note

This operand will make use of any indexes that may be available on the geometries. It is different from other operators that use spatial indexes in that the spatial index is only used when the operator is in the ORDER BY clause.



### Note

Index only kicks in if one of the geometries is a constant e.g. ORDER BY (ST\_GeomFromText('POINT(1 2)') `<<#>>` geom) instead of g1.geom `<<#>>`.

Availability: 2.2.0 -- KNN only available for PostgreSQL 9.1+

## Ver también

`<<->>`, `<#>`

## 7.11 Spatial Relationships

### 7.11.1 Topological Relationships

#### 7.11.1.1 ST\_3DIntersects

**ST\_3DIntersects** — Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)

## Synopsis

boolean **ST\_3DIntersects**( geometry geomA , geometry geomB );

## Description

Overlaps, Touches, Within all imply spatial intersection. If any of the aforementioned returns true, then the geometries also spatially intersect. Disjoint implies false for spatial intersection.



### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs.

Availability: 2.0.0



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



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

Geometry Examples

```
SELECT ST_3DIntersects(pt, line), ST_Intersects(pt, line)
FROM (SELECT 'POINT(0 0 2)::geometry As pt, 'LINESTRING (0 0 1, 0 2 3)::geometry As ↵
 line) As foo;
st_3dintersects | st_intersects
-----+-----
f | t
(1 row)
```

TIN Examples

```
SELECT ST_3DIntersects('TIN(((0 0 0,1 0 0,0 1 0,0 0 0)))::geometry, 'POINT(.1 .1 0):: ↵
 geometry);
st_3dintersects

t
```

See Also

[ST\\_Intersects](#)

7.11.1.2 ST\_Contains

ST\_Contains — Tests if every point of B lies in A, and their interiors have a point in common

Synopsis

boolean **ST\_Contains**(geometry geomA, geometry geomB);

Description

Returns TRUE if geometry A contains geometry B. A contains B if and only if all points of B lie inside (i.e. in the interior or boundary of) A (or equivalently, no points of B lie in the exterior of A), and the interiors of A and B have at least one point in common.

In mathematical terms:  $ST\_Contains(A, B) \Leftrightarrow (A \cap B = B) \wedge (Int(A) \cap Int(B) \neq \emptyset)$

The contains relationship is reflexive: every geometry contains itself. (In contrast, in the [ST\\_ContainsProperly](#) predicate a geometry does *not* properly contain itself.) The relationship is antisymmetric: if  $ST\_Contains(A, B) = \text{true}$  and  $ST\_Contains(B, A) = \text{true}$ , then the two geometries must be topologically equal ( $ST\_Equals(A, B) = \text{true}$ ).

ST\_Contains is the converse of [ST\\_Within](#). So,  $ST\_Contains(A, B) = ST\_Within(B, A)$ .



**Note** Because the interiors must have a common point, a subtlety of the definition is that polygons and lines do *not* contain lines and points lying fully in their boundary. For further details see [Subtleties of OGC Covers, Contains, Within](#). The [ST\\_Covers](#) predicate provides a more inclusive relationship.



**Note** This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Contains`.

Performed by the GEOS module

Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.

**Important**

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

---

**Important**

Do not use this function with invalid geometries. You will get unexpected results.

---

NOTE: this is the "allowable" version that returns a boolean, not an integer.



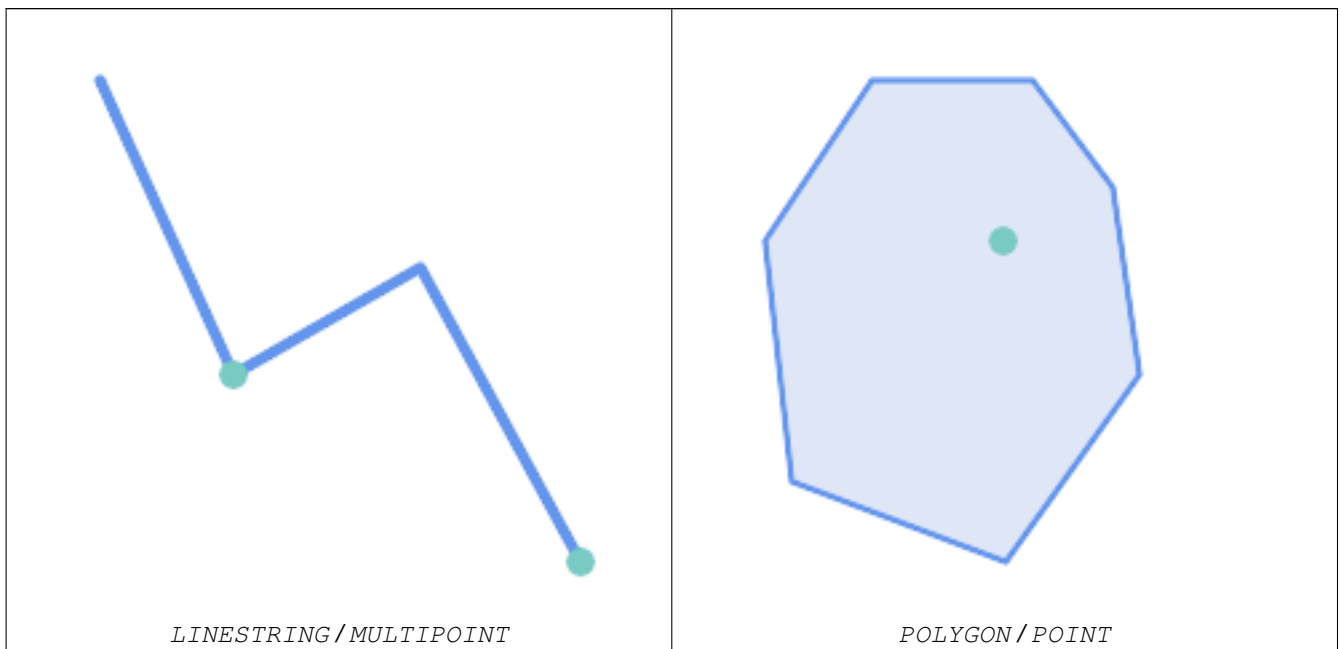
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3 - same as within(geometry B, geometry A)

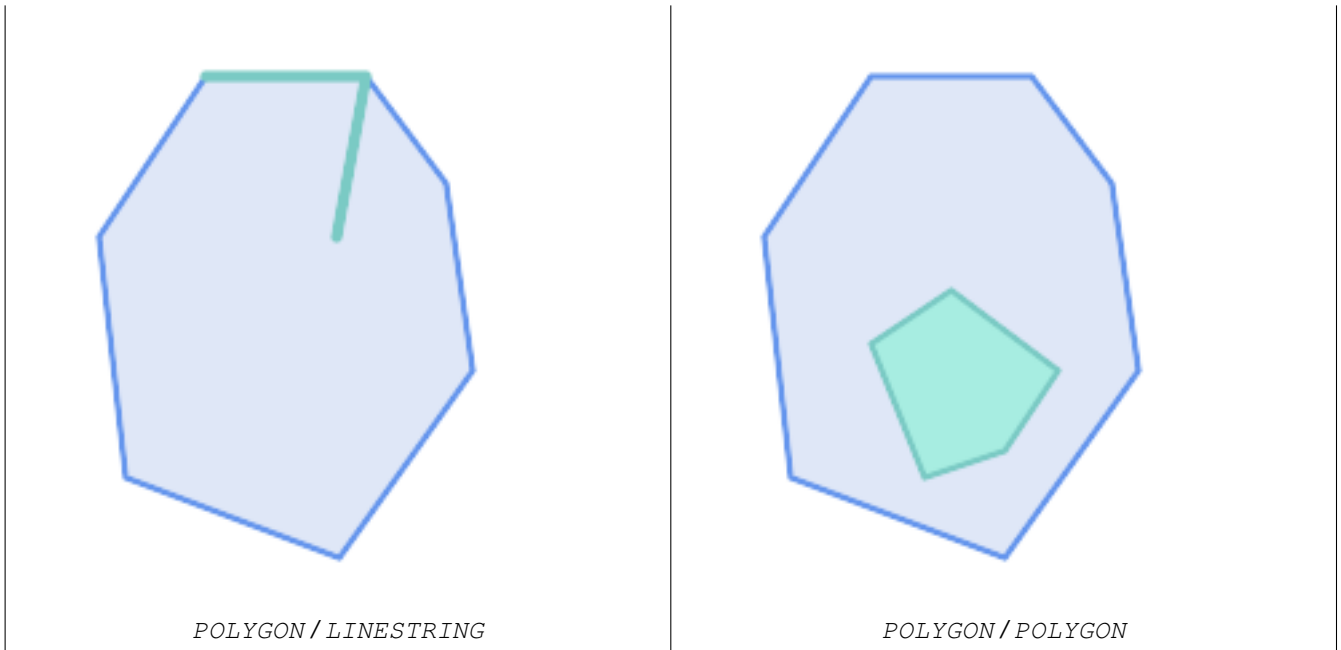


This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

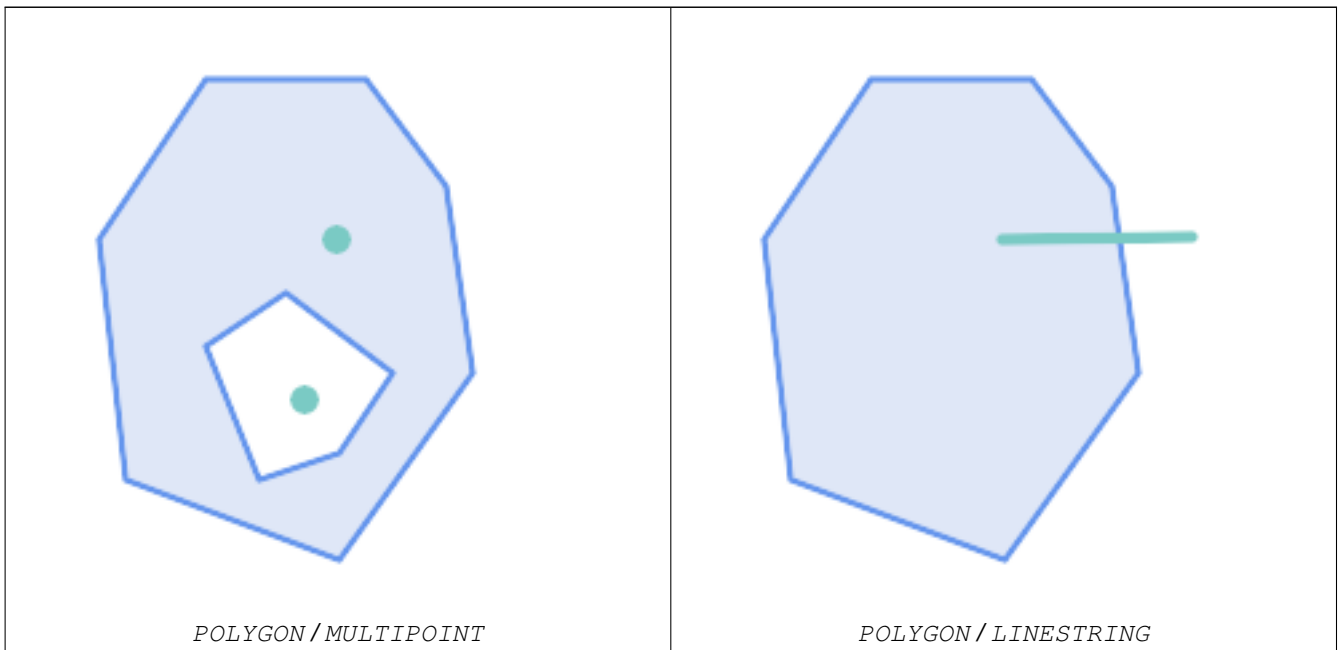
### Examples

ST\_Contains returns TRUE in the following situations:

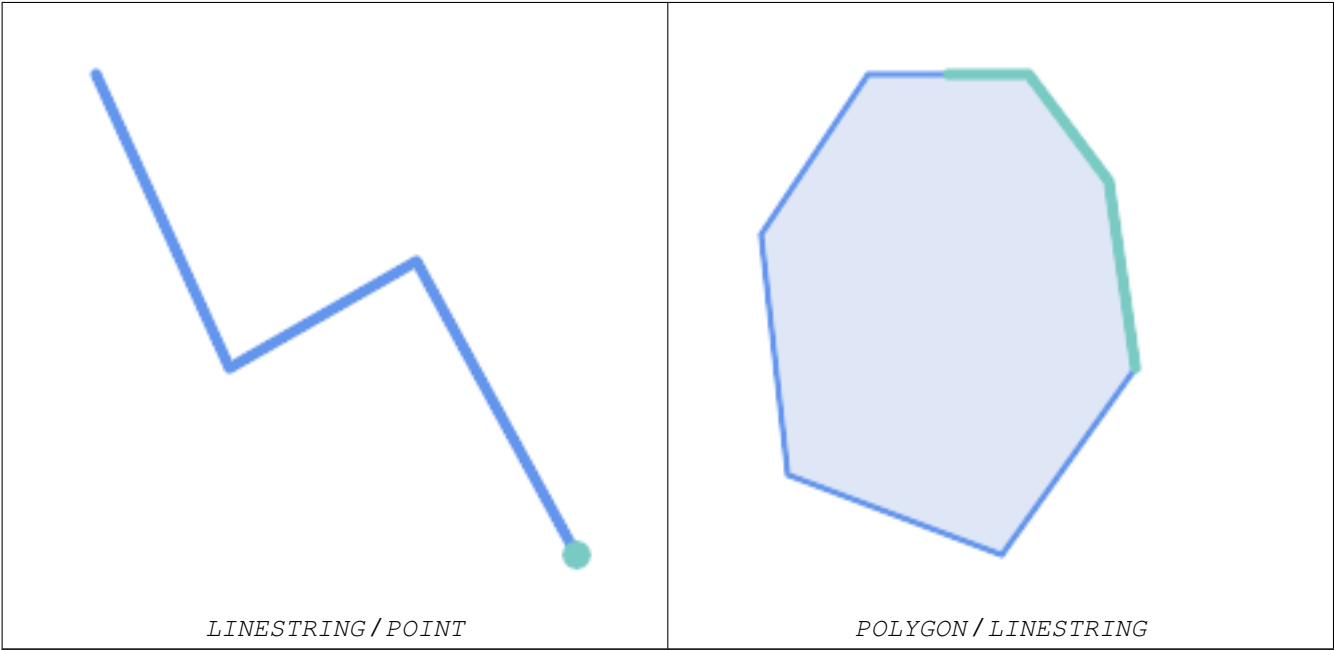




`ST_Contains` returns `FALSE` in the following situations:



Due to the interior intersection condition `ST_Contains` returns `FALSE` in the following situations (whereas `ST_Covers` returns `TRUE`):



```
-- A circle within a circle
SELECT ST_Contains(smallc, bigc) As smallcontainsbig,
 ST_Contains(bigc,smallc) As bigcontainssmall,
 ST_Contains(bigc, ST_Union(smallc, bigc)) as bigcontainsunion,
 ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
 ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
 ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
 ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;

-- Result
smallcontainsbig | bigcontainssmall | bigcontainsunion | bigisunion | bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----+-----+-----
f | t | t | t | t | f

-- Example demonstrating difference between contains and contains properly
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa, ↔
 ST_ContainsProperly(geomA, geomA) AS acontainspropa,
 ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA, ↔
 ST_Boundary(geomA)) As acontainspropba
FROM (VALUES (ST_Buffer(ST_Point(1,1), 5,1)),
 (ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1))),
 (ST_Point(1,1))
) As foo(geomA);

geomtype | acontainsa | acontainspropa | acontainsba | acontainspropba
-----+-----+-----+-----+-----
ST_Polygon | t | f | f | f
ST_LineString | t | f | f | f
ST_Point | t | t | f | f
```

**See Also**

[ST\\_Boundary](#), [ST\\_ContainsProperly](#), [ST\\_Covers](#), [ST\\_CoveredBy](#), [ST\\_Equals](#), [ST\\_Within](#)

**7.11.1.3 ST\_ContainsProperly**

`ST_ContainsProperly` — Tests if every point of B lies in the interior of A

**Synopsis**

```
boolean ST_ContainsProperly(geometry geomA, geometry geomB);
```

**Description**

Returns `true` if every point of B lies in the interior of A (or equivalently, no point of B lies in the the boundary or exterior of A).

In mathematical terms:  $ST\_ContainsProperly(A, B) \Leftrightarrow Int(A) \cap B = B$

A contains B properly if the DE-9IM Intersection Matrix for the two geometries matches `[T**FF*FF*]`

A does not properly contain itself, but does contain itself.

A use for this predicate is computing the intersections of a set of geometries with a large polygonal geometry. Since intersection is a fairly slow operation, it can be more efficient to use `containsProperly` to filter out test geometries which lie fully inside the area. In these cases the intersection is known a priori to be exactly the original test geometry.

**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_ContainsProperly`.

**Note**

The advantage of this predicate over [ST\\_Contains](#) and [ST\\_Intersects](#) is that it can be computed more efficiently, with no need to compute topology at individual points.

Performed by the GEOS module.

Availability: 1.4.0

**Important**

Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

**Important**

Do not use this function with invalid geometries. You will get unexpected results.







**Note**  
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_CoveredBy`.



**Important**  
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`



**Important**  
Do not use this function with invalid geometries. You will get unexpected results.

Performed by the GEOS module

Availability: 1.2.2

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Not an OGC standard, but Oracle has it too.

**Examples**

```
--a circle coveredby a circle
SELECT ST_CoveredBy(smallc,smallc) As smallinsmall,
 ST_CoveredBy(smallc, bigc) As smallcoveredbybig,
 ST_CoveredBy(ST_ExteriorRing(bigc), bigc) As exteriorcoveredbybig,
 ST_Within(ST_ExteriorRing(bigc),bigc) As exeriorwithinbig
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
 ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallinsmall | smallcoveredbybig | exteriorcoveredbybig | exeriorwithinbig
-----+-----+-----+-----
t | t | t | f
(1 row)
```

**See Also**

[ST\\_Contains](#), [ST\\_Covers](#), [ST\\_ExteriorRing](#), [ST\\_Within](#)

**7.11.1.5 ST\_Covers**

`ST_Covers` — Tests if every point of B lies in A

**Synopsis**

boolean `ST_Covers`(geometry geomA, geometry geomB);  
boolean `ST_Covers`(geography geogpolyA, geography geogpointB);

Description

Returns `true` if every point in Geometry/Geography B lies inside (i.e. intersects the interior or boundary of) Geometry/Geography A. Equivalently, tests that no point of B lies outside (in the exterior of) A.

In mathematical terms:  $ST\_Covers(A, B) \Leftrightarrow A \cap B = B$

`ST_Covers` is the converse of `ST_CoveredBy`. So, `ST_Covers(A, B) = ST_CoveredBy(B, A)`.

Generally this function should be used instead of `ST_Contains`, since it has a simpler definition which does not have the quirk that "geometries do not contain their boundary".



**Note**  
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Covers`.



**Important**  
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`



**Important**  
Do not use this function with invalid geometries. You will get unexpected results.

Performed by the GEOS module

Enhanced: 2.4.0 Support for polygon in polygon and line in polygon added for geography type

Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Availability: 1.5 - support for geography was introduced.

Availability: 1.2.2

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Not an OGC standard, but Oracle has it too.

Examples

Geometry example

```
--a circle covering a circle
SELECT ST_Covers(smallc,smallc) As smallinsmall,
 ST_Covers(smallc, bigc) As smallcoversbig,
 ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
 ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
 ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallinsmall | smallcoversbig | bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----
t | f | t | f
(1 row)
```

## Geography Example

```
-- a point with a 300 meter buffer compared to a point, a point and its 10 meter buffer
SELECT ST_Covers(geog_poly, geog_pt) As poly_covers_pt,
 ST_Covers(ST_Buffer(geog_pt,10), geog_pt) As buff_10m_covers_cent
FROM (SELECT ST_Buffer(ST_GeogFromText('SRID=4326;POINT(-99.327 31.4821)'), 300) As
 geog_poly,
 ST_GeogFromText('SRID=4326;POINT(-99.33 31.483)') As geog_pt) As foo;

poly_covers_pt | buff_10m_covers_cent
-----+-----
f | t
```

## See Also

[ST\\_Contains](#), [ST\\_CoveredBy](#), [ST\\_Within](#)

### 7.11.1.6 ST\_Crosses

**ST\_Crosses** — Tests if two geometries have some, but not all, interior points in common

## Synopsis

boolean **ST\_Crosses**(geometry g1, geometry g2);

## Description

Compares two geometry objects and returns `true` if their intersection "spatially crosses"; that is, the geometries have some, but not all interior points in common. The intersection of the interiors of the geometries must be non-empty and must have dimension less than the maximum dimension of the two input geometries, and the intersection of the two geometries must not equal either geometry. Otherwise, it returns `false`. The crosses relation is symmetric and irreflexive.

In mathematical terms:  $ST\_Crosses(A, B) \Leftrightarrow (dim(Int(A) \cap Int(B)) < \max(dim(Int(A)), dim(Int(B))) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$

Geometries cross if their DE-9IM Intersection Matrix matches:

- T\*T\*\*\*\*\* for Point/Line, Point/Area, and Line/Area situations
- T\*\*\*\*\*T\*\* for Line/Point, Area/Point, and Area/Line situations
- 0\*\*\*\*\* for Line/Line situations
- the result is `false` for Point/Point and Area/Area situations



### Note

The OpenGIS Simple Features Specification defines this predicate only for Point/Line, Point/Area, Line/Line, and Line/Area situations. JTS / GEOS extends the definition to apply to Line/Point, Area/Point and Area/Line situations as well. This makes the relation symmetric.



### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.



**Important**

Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`



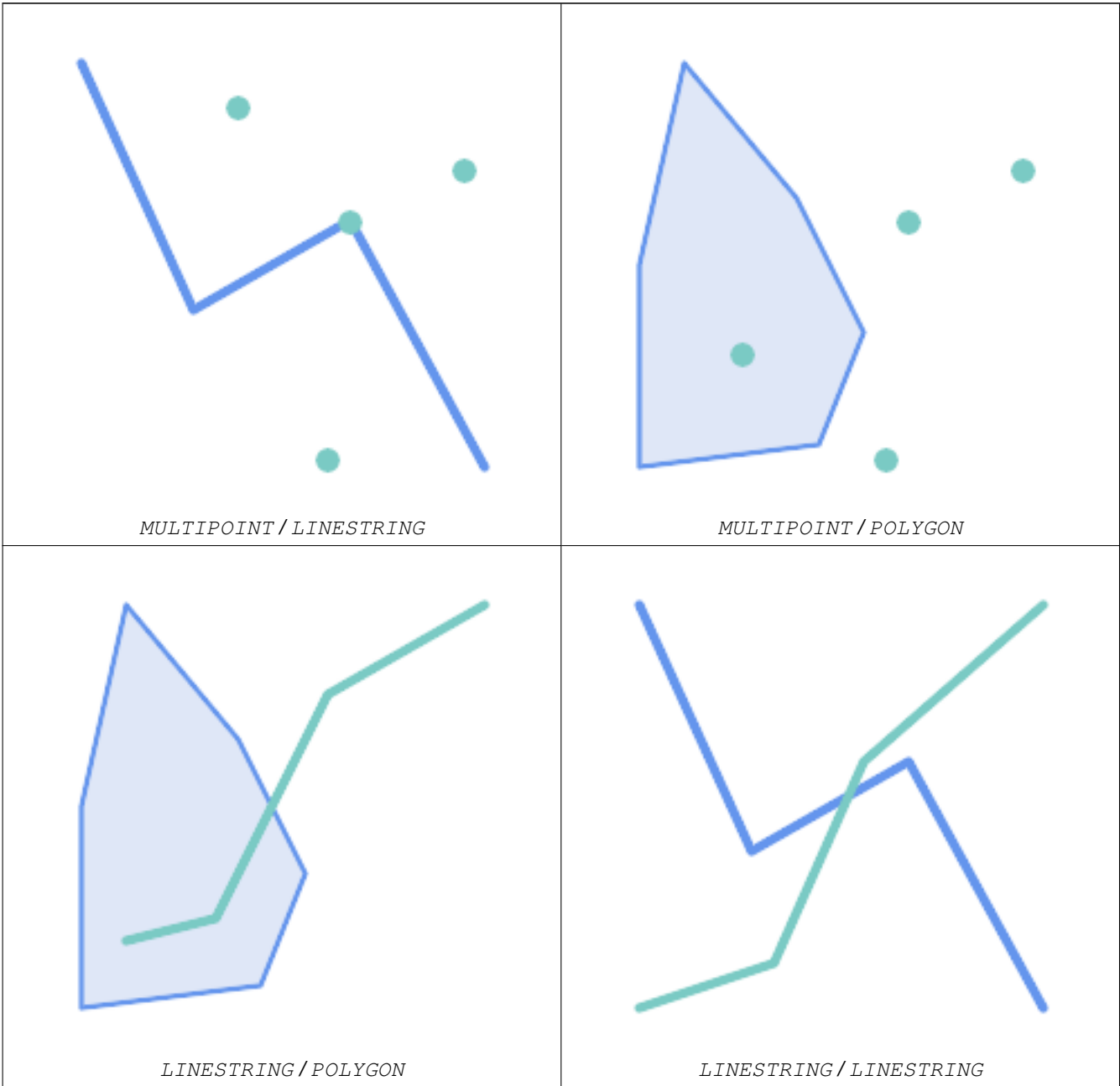
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#), s2.1.13.3



This method implements the SQL/MM specification, SQL-MM 3: 5.1.29

**Examples**

The following situations all return `true`.



Consider a situation where a user has two tables: a table of roads and a table of highways.

```
CREATE TABLE roads (
 id serial NOT NULL,
 geom geometry,
 CONSTRAINT roads_pkey PRIMARY KEY (↵
 road_id)
);
```

```
CREATE TABLE highways (
 id serial NOT NULL,
 the_geom geometry,
 CONSTRAINT roads_pkey PRIMARY KEY (↵
 road_id)
);
```

To determine a list of roads that cross a highway, use a query similar to:

```
SELECT roads.id
FROM roads, highways
WHERE ST_Crosses(roads.geom, highways.geom);
```

### See Also

[ST\\_Contains](#), [ST\\_Overlaps](#)

#### 7.11.1.7 ST\_Disjoint

**ST\_Disjoint** — Tests if two geometries have no points in common

### Synopsis

boolean **ST\_Disjoint**( geometry A , geometry B );

### Description

Returns `true` if two geometries are disjoint. Geometries are disjoint if they have no point in common.

If any other spatial relationship is true for a pair of geometries, they are not disjoint. Disjoint implies that [ST\\_Intersects](#) is false.

In mathematical terms:  $ST\_Disjoint(A, B) \Leftrightarrow A \cap B = \emptyset$



#### Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

Performed by the GEOS module



#### Note

This function call does not use indexes. A negated [ST\\_Intersects](#) predicate can be used as a more performant alternative that uses indexes: `ST_Disjoint(A,B) = NOT ST_Intersects(A,B)`



#### Note

NOTE: this is the "allowable" version that returns a boolean, not an integer.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 //s2.1.13.3 - a.Relate(b, 'FF\*FF\*\*\*\*')



This method implements the SQL/MM specification. SQL-MM 3: 5.1.26

### Examples

```
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING (2 0, 0 2) '::geometry);
st_disjoint

t
(1 row)
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING (0 0, 0 2) '::geometry);
st_disjoint

f
(1 row)
```

### See Also

[ST\\_Intersects](#)

#### 7.11.1.8 ST\_Equals

ST\_Equals — Tests if two geometries include the same set of points

### Synopsis

boolean **ST\_Equals**(geometry A, geometry B);

### Description

Returns `true` if the given geometries are "topologically equal". Use this for a 'better' answer than '='. Topological equality means that the geometries have the same dimension, and their point-sets occupy the same space. This means that the order of vertices may be different in topologically equal geometries. To verify the order of points is consistent use [ST\\_OrderingEquals](#) (it must be noted ST\_OrderingEquals is a little more stringent than simply verifying order of points are the same).

In mathematical terms:  $ST\_Equals(A, B) \Leftrightarrow A = B$

The following relation holds:  $ST\_Equals(A, B) \Leftrightarrow ST\_Within(A,B) \wedge ST\_Within(B,A)$



#### Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2



This method implements the SQL/MM specification. SQL-MM 3: 5.1.24

Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal

## Examples

```
SELECT ST_Equals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
 ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_equals

t
(1 row)

SELECT ST_Equals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
 ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_equals

t
(1 row)
```

## See Also

[ST\\_IsValid](#), [ST\\_OrderingEquals](#), [ST\\_Reverse](#), [ST\\_Within](#)

### 7.11.1.9 ST\_Intersects

**ST\_Intersects** — Tests if two geometries intersect (they have at least one point in common)

## Synopsis

```
boolean ST_Intersects(geometry geomA , geometry geomB);
boolean ST_Intersects(geography geogA , geography geogB);
```

## Description

Returns `true` if two geometries intersect. Geometries intersect if they have any point in common.

For geography, a distance tolerance of 0.00001 meters is used (so points that are very close are considered to intersect).

In mathematical terms:  $ST\_Intersects(A, B) \Leftrightarrow A \cap B \neq \emptyset$

Geometries intersect if their DE-9IM Intersection Matrix matches one of:

- T\*\*\*\*\*
- \*T\*\*\*\*\*
- \*\*\*T\*\*\*\*\*
- \*\*\*\*T\*\*\*\*\*

Spatial intersection is implied by all the other spatial relationship tests, except [ST\\_Disjoint](#), which tests that geometries do NOT intersect.



### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.



Changed: 3.0.0 SFCGAL version removed and native support for 2D TINS added.

Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION.

Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Performed by the GEOS module (for geometry), geography is native

Availability: 1.5 support for geography was introduced.



#### Note

For geography, this function has a distance tolerance of about 0.00001 meters and uses the sphere rather than spheroid calculation.



#### Note

NOTE: this is the "allowable" version that returns a boolean, not an integer.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 //s2.1.13.3 - ST\_Intersects(g1, g2) --> Not (ST\_Disjoint(g1, g2))



This method implements the SQL/MM specification. SQL-MM 3: 5.1.27



This method supports Circular Strings and Curves.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Geometry Examples

```
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING (2 0, 0 2) '::geometry);
st_intersects

f
(1 row)
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING (0 0, 0 2) '::geometry);
st_intersects

t
(1 row)

-- Look up in table. Make sure table has a GiST index on geometry column for faster lookup.
SELECT id, name FROM cities WHERE ST_Intersects(geom, 'SRID=4326;POLYGON((28 53,27.707 ↵
52.293,27 52,26.293 52.293,26 53,26.293 53.707,27 54,27.707 53.707,28 53)) ');
id | name
---+-----
 2 | Minsk
(1 row)
```

## Geography Examples

```
SELECT ST_Intersects (
 'SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568)::geography,
 'SRID=4326;POINT(-43.23456 72.4567772)::geography
);

st_intersects

t
```

**See Also**

[&&](#), [ST\\_3DIntersects](#), [ST\\_Disjoint](#)

**7.11.1.10 ST\_LineCrossingDirection**

**ST\_LineCrossingDirection** — Returns a number indicating the crossing behavior of two LineStrings

**Synopsis**

integer **ST\_LineCrossingDirection**(geometry linestringA, geometry linestringB);

**Description**

Given two linestrings returns an integer between -3 and 3 indicating what kind of crossing behavior exists between them. 0 indicates no crossing. This is only supported for LINESTRINGS.

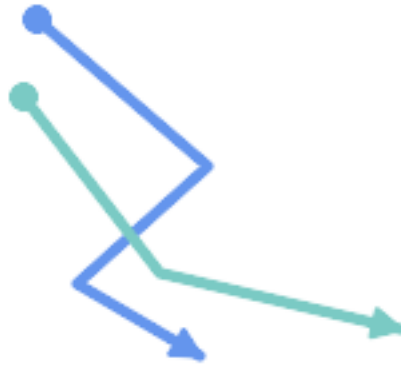
The crossing number has the following meaning:

- 0: LINE NO CROSS
- -1: LINE CROSS LEFT
- 1: LINE CROSS RIGHT
- -2: LINE MULTICROSS END LEFT
- 2: LINE MULTICROSS END RIGHT
- -3: LINE MULTICROSS END SAME FIRST LEFT
- 3: LINE MULTICROSS END SAME FIRST RIGHT

Availability: 1.4

**Examples**

**Example:** LINE CROSS LEFT and LINE CROSS RIGHT

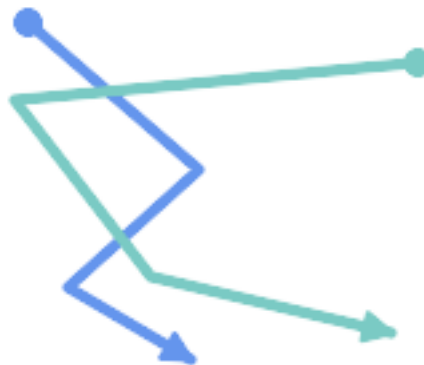


*Blue: Line A; Green: Line B*

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
 ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
 ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
 ST_GeomFromText('LINESTRING (20 140, 71 74, 161 53)') As lineB
) As foo;
```

| A_cross_B | B_cross_A |
|-----------|-----------|
| -1        | 1         |

**Example:** LINE MULTICROSS END SAME FIRST LEFT and LINE MULTICROSS END SAME FIRST RIGHT

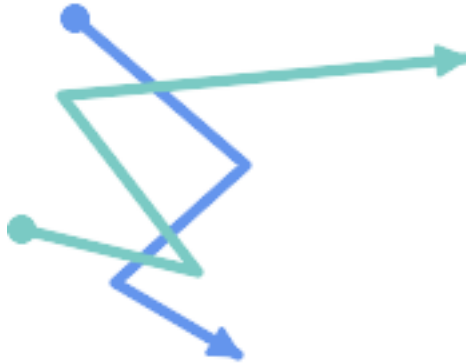


*Blue: Line A; Green: Line B*

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
 ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
 ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
 ST_GeomFromText('LINESTRING(171 154,20 140,71 74,161 53)') As lineB
) As foo;
```

| A_cross_B | B_cross_A |
|-----------|-----------|
| 3         | -3        |

**Example:** LINE\_MULTICROSS\_END\_LEFT and LINE\_MULTICROSS\_END\_RIGHT



*Blue: Line A; Green: Line B*

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
 ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
 ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
 ST_GeomFromText('LINESTRING(5 90, 71 74, 20 140, 171 154)') As lineB
) As foo;
```

| A_cross_B | B_cross_A |
|-----------|-----------|
| -2        | 2         |

**Example:** Finds all streets that cross

```
SELECT s1.gid, s2.gid, ST_LineCrossingDirection(s1.geom, s2.geom)
FROM streets s1 CROSS JOIN streets s2
 ON (s1.gid != s2.gid AND s1.geom && s2.geom)
WHERE ST_LineCrossingDirection(s1.geom, s2.geom)
> 0;
```

## See Also

[ST\\_Crosses](#)

### 7.11.1.11 ST\_OrderingEquals

ST\_OrderingEquals — Tests if two geometries represent the same geometry and have points in the same directional order

## Synopsis

boolean **ST\_OrderingEquals**(geometry A, geometry B);

## Description

`ST_OrderingEquals` compares two geometries and returns t (TRUE) if the geometries are equal and the coordinates are in the same order; otherwise it returns f (FALSE).



### Note

This function is implemented as per the ArcSDE SQL specification rather than SQL-MM. [http://edndoc.esri.com/arcsde/9.1/sql\\_api/sqlapi3.htm#ST\\_OrderingEquals](http://edndoc.esri.com/arcsde/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals)



This method implements the SQL/MM specification. SQL-MM 3: 5.1.43

## Examples

```
SELECT ST_OrderingEquals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
 ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_orderingequals

f
(1 row)

SELECT ST_OrderingEquals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
 ST_GeomFromText('LINESTRING(0 0, 0 0, 10 10)'));
 st_orderingequals

t
(1 row)

SELECT ST_OrderingEquals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
 ST_GeomFromText('LINESTRING(0 0, 0 0, 10 10)'));
 st_orderingequals

f
(1 row)
```

## See Also

[&&](#), [ST\\_Equals](#), [ST\\_Reverse](#)

### 7.11.1.12 ST\_Overlaps

`ST_Overlaps` — Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other

## Synopsis

boolean **ST\_Overlaps**(geometry A, geometry B);

## Description

Returns TRUE if geometry A and B "spatially overlap". Two geometries overlap if they have the same dimension, their interiors intersect in that dimension. and each has at least one point inside the other (or equivalently, neither one covers the other). The overlaps relation is symmetric and irreflexive.

In mathematical terms:  $ST\_Overlaps(A, B) \Leftrightarrow (dim(A) = dim(B) = dim(Int(A) \cap Int(B))) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$



**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Overlaps`.

Performed by the GEOS module



**Important**

Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

NOTE: this is the "allowable" version that returns a boolean, not an integer.



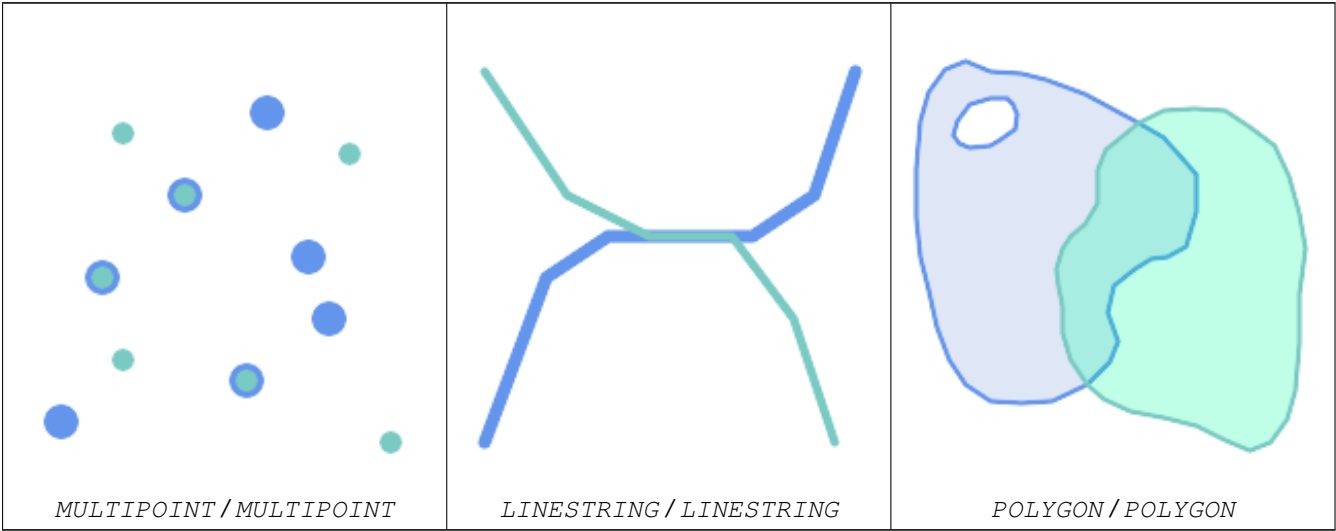
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3

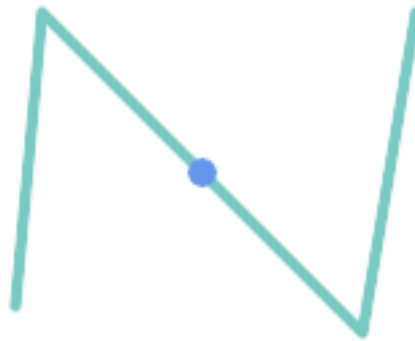


This method implements the SQL/MM specification. SQL-MM 3: 5.1.32

**Examples**

`ST_Overlaps` returns TRUE in the following situations:





A Point on a LineString is contained, but since it has lower dimension it does not overlap or cross.

```
SELECT ST_Overlaps(a,b) AS overlaps, ST_Crosses(a,b) AS crosses,
 ST_Intersects(a, b) AS intersects, ST_Contains(b,a) AS b_contains_a
FROM (SELECT ST_GeomFromText('POINT (100 100)') As a,
 ST_GeomFromText('LINESTRING (30 50, 40 160, 160 40, 180 160)') AS b) AS t
```

| overlaps | crosses | intersects | b_contains_a |
|----------|---------|------------|--------------|
| f        | f       | t          | t            |



A LineString that partly covers a Polygon intersects and crosses, but does not overlap since it has different dimension.

```
SELECT ST_Overlaps(a,b) AS overlaps, ST_Crosses(a,b) AS crosses,
 ST_Intersects(a, b) AS intersects, ST_Contains(a,b) AS contains
FROM (SELECT ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
 ST_GeomFromText('LINESTRING(10 10, 190 190)') AS b) AS t;
```

| overlap | crosses | intersects | contains |
|---------|---------|------------|----------|
| f       | t       | t          | f        |



Two Polygons that intersect but with neither contained by the other overlap, but do not cross because their intersection has the same dimension.

```
SELECT ST_Overlaps(a,b) AS overlaps, ST_Crosses(a,b) AS crosses,
 ST_Intersects(a, b) AS intersects, ST_Contains(b, a) AS b_contains_a,
 ST_Dimension(a) AS dim_a, ST_Dimension(b) AS dim_b,
 ST_Dimension(ST_Intersection(a,b)) AS dim_int
FROM (SELECT ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
 ST_GeomFromText('POLYGON ((110 180, 20 60, 130 90, 110 180))') AS b) AS t;
```

| overlaps | crosses | intersects | b_contains_a | dim_a | dim_b | dim_int |
|----------|---------|------------|--------------|-------|-------|---------|
| t        | f       | t          | f            | 2     | 2     | 2       |

See Also

[ST\\_Contains](#), [ST\\_Crosses](#), [ST\\_Dimension](#), [ST\\_Intersects](#)

7.11.1.13 ST\_Relate

ST\_Relate — Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix

Synopsis

boolean **ST\_Relate**(geometry geomA, geometry geomB, text intersectionMatrixPattern);  
text **ST\_Relate**(geometry geomA, geometry geomB);  
text **ST\_Relate**(geometry geomA, geometry geomB, integer boundaryNodeRule);

Description

These functions allow testing and evaluating the spatial (topological) relationship between two geometries, as defined by the [Dimensionally Extended 9-Intersection Model](#) (DE-9IM).  
The DE-9IM is specified as a 9-element matrix indicating the dimension of the intersections between the Interior, Boundary and Exterior of two geometries. It is represented by a 9-character text string using the symbols 'F', '0', '1', '2' (e.g. 'FF1FF0102').  
A specific kind of spatial relationship can be tested by matching the intersection matrix to an *intersection matrix pattern*. Patterns can include the additional symbols 'T' (meaning "intersection is non-empty") and '\*' (meaning "any value"). Common spatial



relationships are provided by the named functions **ST\_Contains**, **ST\_ContainsProperly**, **ST\_Covers**, **ST\_CoveredBy**, **ST\_Crosses**, **ST\_Disjoint**, **ST\_Equals**, **ST\_Intersects**, **ST\_Overlaps**, **ST\_Touches**, and **ST\_Within**. Using an explicit pattern allows testing multiple conditions of intersects, crosses, etc in one step. It also allows testing spatial relationships which do not have a named spatial relationship function. For example, the relationship "Interior-Intersects" has the DE-9IM pattern `T*****`, which is not evaluated by any named predicate.

For more information refer to Section 5.1.

**Variant 1:** Tests if two geometries are spatially related according to the given `intersectionMatrixPattern`.



#### Note

Unlike most of the named spatial relationship predicates, this does NOT automatically include an index call. The reason is that some relationships are true for geometries which do NOT intersect (e.g. Disjoint). If you are using a relationship pattern that requires intersection, then include the `&&` index call.



#### Note

It is better to use a named relationship function if available, since they automatically use a spatial index where one exists. Also, they may implement performance optimizations which are not available with full relate evaluation.

**Variant 2:** Returns the DE-9IM matrix string for the spatial relationship between the two input geometries. The matrix string can be tested for matching a DE-9IM pattern using **ST\_RelateMatch**.

**Variant 3:** Like variant 2, but allows specifying a **Boundary Node Rule**. A boundary node rule allows finer control over whether the endpoints of MultiLineStrings are considered to lie in the DE-9IM Interior or Boundary. The `boundaryNodeRule` values are:

- 1: **OGC-Mod2** - line endpoints are in the Boundary if they occur an odd number of times. This is the rule defined by the OGC SFS standard, and is the default for `ST_Relate`.
- 2: **Endpoint** - all endpoints are in the Boundary.
- 3: **MultivalentEndpoint** - endpoints are in the Boundary if they occur more than once. In other words, the boundary is all the "attached" or "inner" endpoints (but not the "unattached/outer" ones).
- 4: **MonovalentEndpoint** - endpoints are in the Boundary if they occur only once. In other words, the boundary is all the "unattached" or "outer" endpoints.

This function is not in the OGC spec, but is implied. see s2.1.13.2



This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**. s2.1.1.2 // s2.1.13.3



This method implements the SQL/MM specification. SQL-MM 3: 5.1.25

Performed by the GEOS module

Enhanced: 2.0.0 - added support for specifying boundary node rule.



#### Important

Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

## Examples

Using the boolean-valued function to test spatial relationships.

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer('POINT(1 2)', 2), '0FFFFFF212');
st_relate

t

SELECT ST_Relate(POINT(1 2)', ST_Buffer('POINT(1 2)', 2), '*FF*FF212');
st_relate

t
```

Testing a custom spatial relationship pattern as a query condition, with && to enable using a spatial index.

```
-- Find compounds that properly intersect (not just touch) a poly (Interior Intersects)

SELECT c.* , p.name As poly_name
FROM polys AS p
INNER JOIN compounds As c
 ON c.geom && p.geom
 AND ST_Relate(p.geom, c.geom, 'T*****');
```

Computing the intersection matrix for spatial relationships.

```
SELECT ST_Relate('POINT(1 2)',
 ST_Buffer('POINT(1 2)', 2));

0FFFFFF212

SELECT ST_Relate('LINESTRING(1 2, 3 4)',
 'LINESTRING(5 6, 7 8)');

FF1FF0102
```

Using different Boundary Node Rules to compute the spatial relationship between a LineString and a MultiLineString with a duplicate endpoint (3 3):

- Using the **OGC-Mod2** rule (1) the duplicate endpoint is in the **interior** of the MultiLineString, so the DE-9IM matrix entry [aB:bI] is 0 and [aB:bB] is F.
- Using the **Endpoint** rule (2) the duplicate endpoint is in the **boundary** of the MultiLineString, so the DE-9IM matrix entry [aB:bI] is F and [aB:bB] is 0.

```
WITH data AS (SELECT
 'LINESTRING(1 1, 3 3)::geometry AS a_line,
 'MULTILINESTRING((3 3, 3 5), (3 3, 5 3)):: geometry AS b_multiline
)
SELECT ST_Relate(a_line, b_multiline, 1) AS bnr_mod2,
 ST_Relate(a_line, b_multiline, 2) AS bnr_endpoint
FROM data;

bnr_mod2 | bnr_endpoint
-----+-----
FF10F0102 | FF1F00102
```

## See Also

Section 5.1, [ST\\_RelateMatch](#), [ST\\_Contains](#), [ST\\_ContainsProperly](#), [ST\\_Covers](#), [ST\\_CoveredBy](#), [ST\\_Crosses](#), [ST\\_Disjoint](#), [ST\\_Equals](#), [ST\\_Intersects](#), [ST\\_Overlaps](#), [ST\\_Touches](#), [ST\\_Within](#)

7.11.1.14 ST\_RelateMatch

ST\_RelateMatch — Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern

Synopsis

boolean **ST\_RelateMatch**(text intersectionMatrix, text intersectionMatrixPattern);

Description

Tests if a **Dimensionally Extended 9-Intersection Model** (DE-9IM) intersectionMatrix value satisfies an intersectionMatrixPattern. Intersection matrix values can be computed by **ST\_Relate**.

For more information refer to Section 5.1.

Performed by the GEOS module

Availability: 2.0.0

Examples

```
SELECT ST_RelateMatch('101202FFF', 'TTTTTFFFF') ;
-- result --
t
```

Patterns for common spatial relationships matched against intersection matrix values, for a line in various positions relative to a polygon

```
SELECT pat.name AS relationship, pat.val AS pattern,
 mat.name AS position, mat.val AS matrix,
 ST_RelateMatch(mat.val, pat.val) AS match
FROM (VALUES ('Equality', 'T1FF1FFF1'),
 ('Overlaps', 'T*T***T**'),
 ('Within', 'T*F**F***'),
 ('Disjoint', 'FF*FF****')) AS pat(name,val)
CROSS JOIN
 (VALUES ('non-intersecting', 'FF1FF0212'),
 ('overlapping', '1010F0212'),
 ('inside', '1FF0FF212')) AS mat(name,val);
```

| relationship | pattern   | position         | matrix    | match |
|--------------|-----------|------------------|-----------|-------|
| Equality     | T1FF1FFF1 | non-intersecting | FF1FF0212 | f     |
| Equality     | T1FF1FFF1 | overlapping      | 1010F0212 | f     |
| Equality     | T1FF1FFF1 | inside           | 1FF0FF212 | f     |
| Overlaps     | T*T***T** | non-intersecting | FF1FF0212 | f     |
| Overlaps     | T*T***T** | overlapping      | 1010F0212 | t     |
| Overlaps     | T*T***T** | inside           | 1FF0FF212 | f     |
| Within       | T*F**F*** | non-intersecting | FF1FF0212 | f     |
| Within       | T*F**F*** | overlapping      | 1010F0212 | f     |
| Within       | T*F**F*** | inside           | 1FF0FF212 | t     |
| Disjoint     | FF*FF**** | non-intersecting | FF1FF0212 | t     |
| Disjoint     | FF*FF**** | overlapping      | 1010F0212 | f     |
| Disjoint     | FF*FF**** | inside           | 1FF0FF212 | f     |

See Also

Section 5.1, **ST\_Relate**

7.11.1.15 ST\_Touches

ST\_Touches — Tests if two geometries have at least one point in common, but their interiors do not intersect

Synopsis

boolean **ST\_Touches**(geometry A, geometry B);

Description

Returns TRUE if A and B intersect, but their interiors do not intersect. Equivalently, A and B have at least one point in common, and the common points lie in at least one boundary. For Point/Point inputs the relationship is always FALSE, since points do not have a boundary.

In mathematical terms:  $ST\_Touches(A, B) \Leftrightarrow (Int(A) \cap Int(B) \neq \emptyset) \wedge (A \cap B \neq \emptyset)$

This relationship holds if the DE-9IM Intersection Matrix for the two geometries matches one of:

- FT\*\*\*\*\*
- F\*\*T\*\*\*\*\*
- F\*\*\*T\*\*\*\*



**Note**  
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid using an index, use `_ST_Touches` instead.



**Important**  
Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3

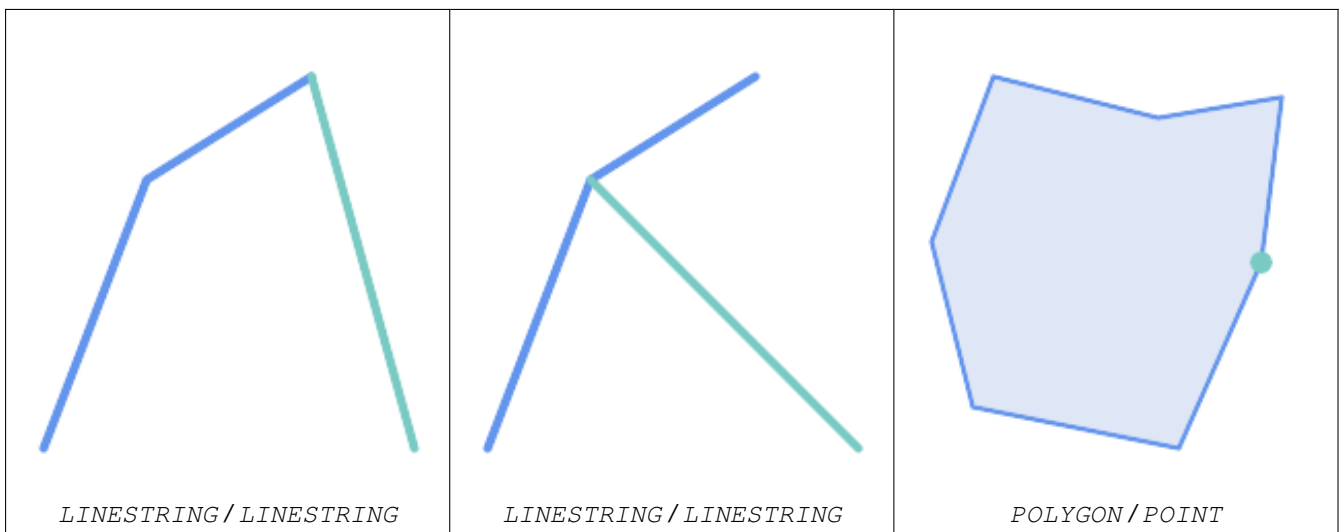


This method implements the SQL/MM specification. SQL-MM 3: 5.1.28

Examples

The ST\_Touches predicate returns TRUE in the following examples.

|                   |                   |                      |
|-------------------|-------------------|----------------------|
|                   |                   |                      |
| POLYGON / POLYGON | POLYGON / POLYGON | POLYGON / LINESTRING |



```
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(1 1)::geometry');
st_touches

f
(1 row)

SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(0 2)::geometry');
st_touches

t
(1 row)
```

#### 7.11.1.16 ST\_Within

**ST\_Within** — Tests if every point of A lies in B, and their interiors have a point in common

##### Synopsis

boolean **ST\_Within**(geometry A, geometry B);

##### Description

Returns TRUE if geometry A is within geometry B. A is within B if and only if all points of A lie inside (i.e. in the interior or boundary of) B (or equivalently, no points of A lie in the exterior of B), and the interiors of A and B have at least one point in common.

For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.

In mathematical terms:  $ST\_Within(A, B) \Leftrightarrow (A \cap B = A) \wedge (Int(A) \cap Int(B) \neq \emptyset)$

The within relation is reflexive: every geometry is within itself. The relation is antisymmetric: if  $ST\_Within(A, B) = \text{true}$  and  $ST\_Within(B, A) = \text{true}$ , then the two geometries must be topologically equal ( $ST\_Equals(A, B) = \text{true}$ ).

**ST\_Within** is the converse of **ST\_Contains**. So,  $ST\_Within(A, B) = ST\_Contains(B, A)$ .



##### Note

Because the interiors must have a common point, a subtlety of the definition is that lines and points lying fully in the boundary of polygons or lines are *not* within the geometry. For further details see [Subtleties of OGC Covers, Contains, Within](#). The **ST\_CoveredBy** predicate provides a more inclusive relationship.



**Note**  
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Within`.

Performed by the GEOS module

Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.



**Important**  
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`



**Important**  
Do not use this function with invalid geometries. You will get unexpected results.

- NOTE: this is the "allowable" version that returns a boolean, not an integer.
- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3 - a.Relate(b, 'T\*\*F\*\*F\*\*\*')
  - ✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.30

Examples

```
--a circle within a circle
SELECT ST_Within(smallc,smallc) As smallinsmall,
 ST_Within(smallc, bigc) As smallinbig,
 ST_Within(bigc,smallc) As biginsmall,
 ST_Within(ST_Union(smallc, bigc), bigc) as unioninbig,
 ST_Within(bigc, ST_Union(smallc, bigc)) as beginunion,
 ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion
FROM
(
SELECT ST_Buffer(ST_GeomFromText('POINT(50 50)'), 20) As smallc,
 ST_Buffer(ST_GeomFromText('POINT(50 50)'), 40) As bigc) As foo;
--Result
smallinsmall | smallinbig | biginsmall | unioninbig | beginunion | bigisunion
-----+-----+-----+-----+-----+-----
t | t | f | t | t | t
(1 row)
```

**See Also**

[ST\\_Contains](#), [ST\\_CoveredBy](#), [ST\\_Equals](#), [ST\\_IsValid](#)

**7.11.2 Distance Relationships****7.11.2.1 ST\_3DDWithin**

**ST\_3DDWithin** — Tests if two 3D geometries are within a given 3D distance

**Synopsis**

boolean **ST\_3DDWithin**(geometry g1, geometry g2, double precision distance\_of\_srid);

**Description**

Returns true if the 3D distance between two geometry values is no larger than distance `distance_of_srid`. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense the source geometries must be in the same coordinate system (have the same SRID).

**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.



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



This function supports Polyhedral surfaces.



This method implements the SQL/MM specification. SQL-MM ?

Availability: 2.0.0

Examples

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point ←
and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ←
units as final.
SELECT ST_3DDWithin(
 ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)'),2163),
 ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ←
 20)'),2163),
 126.8
) As within_dist_3d,
ST_DWithin(
 ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)'),2163),
 ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ←
 20)'),2163),
 126.8
) As within_dist_2d;

within_dist_3d | within_dist_2d
-----+-----
f | t
```

See Also

[ST\\_3DDFullyWithin](#), [ST\\_DWithin](#), [ST\\_DFullyWithin](#), [ST\\_3DDDistance](#), [ST\\_Distance](#), [ST\\_3DMaxDistance](#), [ST\\_Transform](#)

7.11.2.2 ST\_3DDFullyWithin

ST\_3DDFullyWithin — Tests if two 3D geometries are entirely within a given 3D distance

Synopsis

boolean **ST\_3DDFullyWithin**(geometry g1, geometry g2, double precision distance);

Description

Returns true if the 3D geometries are fully within the specified distance of one another. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.



**Note**  
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Availability: 2.0.0



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



This function supports Polyhedral surfaces.



Examples

```
-- This compares the difference between fully within and distance within as well
-- as the distance fully within for the 2D footprint of the line/point vs. the 3d fully
 within
 SELECT ST_3DDFullyWithin(geom_a, geom_b, 10) as D3DFullyWithin10, ST_3DDWithin(geom_a,
 geom_b, 10) as D3DWithin10,
 ST_DFullyWithin(geom_a, geom_b, 20) as D2DFullyWithin20,
 ST_3DDFullyWithin(geom_a, geom_b, 20) as D3DFullyWithin20 from
 (select ST_GeomFromEWKT('POINT(1 1 2)') as geom_a,
 ST_GeomFromEWKT('LINESTRING(1 5 2, 2 7 20, 1 9 100, 14 12 3)') as geom_b) t1;
d3dfullywithin10 | d3dwithin10 | d2dfullywithin20 | d3dfullywithin20
-----+-----+-----+-----
f | t | t | f
```

See Also

[ST\\_3DDWithin](#), [ST\\_DWithin](#), [ST\\_DFullyWithin](#), [ST\\_3DMaxDistance](#)

7.11.2.3 ST\_DFullyWithin

ST\_DFullyWithin — Tests if two geometries are entirely within a given distance

Synopsis

boolean **ST\_DFullyWithin**(geometry g1, geometry g2, double precision distance);

Description

Returns true if the geometries are entirely within the specified distance of one another. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.



**Note**  
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Availability: 1.5.0

Examples

```
postgis=# SELECT ST_DFullyWithin(geom_a, geom_b, 10) as DFullyWithin10, ST_DWithin(geom_a,
 geom_b, 10) as DWithin10, ST_DFullyWithin(geom_a, geom_b, 20) as DFullyWithin20 from
 (select ST_GeomFromText('POINT(1 1)') as geom_a, ST_GeomFromText('LINESTRING(1 5, 2 7, 1
 9, 14 12)') as geom_b) t1;

DFullyWithin10 | DWithin10 | DFullyWithin20 |
-----+-----+-----+
f | t | t |
```

**See Also**

[ST\\_MaxDistance](#), [ST\\_DWithin](#), [ST\\_3DDWithin](#), [ST\\_3DDFullyWithin](#)

**7.11.2.4 ST\_DWithin**

**ST\_DWithin** — Tests if two geometries are within a given distance

**Synopsis**

```
boolean ST_DWithin(geometry g1, geometry g2, double precision distance_of_srid);
boolean ST_DWithin(geography gg1, geography gg2, double precision distance_meters, boolean use_spheroid = true);
```

**Description**

Returns true if the geometries are within a given distance

For geometry: The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must be in the same coordinate system (have the same SRID).

For geography: units are in meters and distance measurement defaults to `use_spheroid = true`. For faster evaluation use `use_spheroid = false` to measure on the sphere.

**Note**

Use [ST\\_3DDWithin](#) for 3D geometries.

**Note**

This function call includes a bounding box comparison that makes use of any indexes that are available on the geometries.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).

Availability: 1.5.0 support for geography was introduced

Enhanced: 2.1.0 improved speed for geography. See [Making Geography faster](#) for details.

Enhanced: 2.1.0 support for curved geometries was introduced.

Prior to 1.3, [ST\\_Expand](#) was commonly used in conjunction with `&&` and `ST_Distance` to test for distance, and in pre-1.3.4 this function used that logic. From 1.3.4, `ST_DWithin` uses a faster short-circuit distance function.

**Examples**

```
-- Find the nearest hospital to each school
-- that is within 3000 units of the school.
-- We do an ST_DWithin search to utilize indexes to limit our search list
-- that the non-indexable ST_Distance needs to process
-- If the units of the spatial reference is meters then units would be meters
SELECT DISTINCT ON (s.gid) s.gid, s.school_name, s.geom, h.hospital_name
FROM schools s
LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
ORDER BY s.gid, ST_Distance(s.geom, h.geom);
```

```
-- The schools with no close hospitals
-- Find all schools with no hospital within 3000 units
-- away from the school. Units is in units of spatial ref (e.g. meters, feet, degrees)
SELECT s.gid, s.school_name
FROM schools s
 LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
WHERE h.gid IS NULL;

-- Find broadcasting towers that receiver with limited range can receive.
-- Data is geometry in Spherical Mercator (SRID=3857), ranges are approximate.

-- Create geometry index that will check proximity limit of user to tower
CREATE INDEX ON broadcasting_towers using gist (geom);

-- Create geometry index that will check proximity limit of tower to user
CREATE INDEX ON broadcasting_towers using gist (ST_Expand(geom, sending_range));

-- Query towers that 4-kilometer receiver in Minsk Hackerspace can get
-- Note: two conditions, because shorter LEAST(b.sending_range, 4000) will not use index.
SELECT b.tower_id, b.geom
FROM broadcasting_towers b
WHERE ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', 4000)
 AND ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', b.sending_range);
```

## See Also

[ST\\_Distance](#), [ST\\_3DDWithin](#)

### 7.11.2.5 ST\_PointInsideCircle

**ST\_PointInsideCircle** — Tests if a point geometry is inside a circle defined by a center and radius

## Synopsis

boolean **ST\_PointInsideCircle**(geometry a\_point, float center\_x, float center\_y, float radius);

## Description

Returns true if the geometry is a point and is inside the circle with center `center_x`, `center_y` and radius `radius`.



### Warning

Does not use spatial indexes. Use [ST\\_DWithin](#) instead.

Availability: 1.2

Changed: 2.2.0 In prior versions this was called `ST_Point_Inside_Circle`

## Examples

```
SELECT ST_PointInsideCircle(ST_Point(1,2), 0.5, 2, 3);
 st_pointinsidecircle

t
```

**See Also**[ST\\_DWithin](#)

## 7.12 Measurement Functions

### 7.12.1 ST\_Area

ST\_Area — Returns the area of a polygonal geometry.

**Synopsis**

```
float ST_Area(geometry g1);
float ST_Area(geography geog, boolean use_spheroid = true);
```

**Descripción**

Returns the area of a polygonal geometry. For geometry types a 2D Cartesian (planar) area is computed, with units specified by the SRID. For geography types by default area is determined on a spheroid with units in square meters. To compute the area using the faster but less accurate spherical model use `ST_Area(geog, false)`.

Enhanced: 2.0.0 - support for 2D polyhedral surfaces was introduced.

Enhanced: 2.2.0 - measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires PROJ >= 4.9.0 to take advantage of the new feature.

Changed: 3.0.0 - does not depend on SFCGAL anymore.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 8.1.2, 9.5.3



This function supports Polyhedral surfaces.

**Note**

For polyhedral surfaces, only supports 2D polyhedral surfaces (not 2.5D). For 2.5D, may give a non-zero answer, but only for the faces that sit completely in XY plane.

**Ejemplos**

Return area in square feet for a plot of Massachusetts land and multiply by conversion to get square meters. Note this is in square feet because EPSG:2249 is Massachusetts State Plane Feet

```
select ST_Area(geom) sqft,
 ST_Area(geom) * 0.3048 ^ 2 sqm
from (
 select 'SRID=2249;POLYGON((743238 2967416,743238 2967450,
 743265 2967450,743265.625 2967416,743238 2967416))' ::
 geometry geom
) subquery;
sqft sqm
928.625 86.27208552
2514 2534
```



Descripción

Returns the azimuth in radians of the target point from the origin point, or NULL if the two points are coincident. The azimuth angle is a positive clockwise angle referenced from the positive Y axis (geometry) or the North meridian (geography): North = 0; Northeast =  $\pi/4$ ; East =  $\pi/2$ ; Southeast =  $3\pi/4$ ; South =  $\pi$ ; Southwest  $5\pi/4$ ; West =  $3\pi/2$ ; Northwest =  $7\pi/4$ .

For the geography type, the azimuth solution is known as the **inverse geodesic problem**.

The azimuth is a mathematical concept defined as the angle between a reference vector and a point, with angular units in radians. The result value in radians can be converted to degrees using the PostgreSQL function `degrees()`.

Azimuth can be used in conjunction with **ST\_Translate** to shift an object along its perpendicular axis. See the `upgis_lineshift()` function in the **PostGIS wiki** for an implementation of this.

Disponibilidad: 1.1.0

Enhanced: 2.0.0 support for geography was introduced.

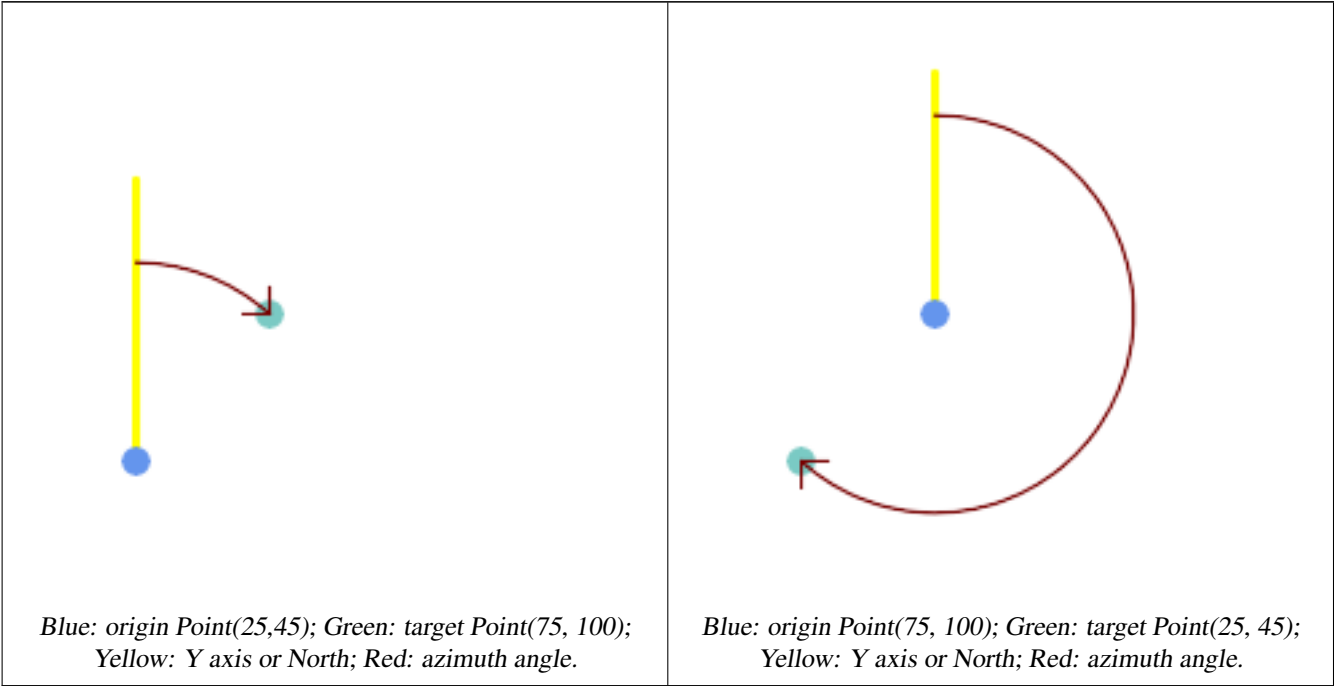
Enhanced: 2.2.0 measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires PROJ >= 4.9.0 to take advantage of the new feature.

Ejemplos

Geometry Azimuth in degrees

```
SELECT degrees(ST_Azimuth(ST_Point(25, 45), ST_Point(75, 100))) AS degA_B,
 degrees(ST_Azimuth(ST_Point(75, 100), ST_Point(25, 45))) AS degB_A;
```

| degA_B           | degB_A           |
|------------------|------------------|
| 42.2736890060937 | 222.273689006094 |



Ver también

[ST\\_Angle](#), [ST\\_Point](#), [ST\\_Translate](#), [ST\\_Project](#), [PostgreSQL Math Functions](#)

### 7.12.3 ST\_Angle

ST\_Angle — Returns the angle between two vectors defined by 3 or 4 points, or 2 lines.

#### Synopsis

```
float ST_Angle(geometry point1, geometry point2, geometry point3, geometry point4);
float ST_Angle(geometry line1, geometry line2);
```

#### Descripción

Computes the clockwise angle between two vectors.

**Variant 1:** computes the angle enclosed by the points P1-P2-P3. If a 4th point provided computes the angle points P1-P2 and P3-P4

**Variant 2:** computes the angle between two vectors S1-E1 and S2-E2, defined by the start and end points of the input lines

The result is a positive angle between 0 and  $2\pi$  radians. The radian result can be converted to degrees using the PostgreSQL function `degrees()`.

Note that `ST_Angle(P1,P2,P3) = ST_Angle(P2,P1,P2,P3)`.

Availability: 2.5.0

#### Ejemplos

Angle between three points

```
SELECT degrees(ST_Angle('POINT(0 0)', 'POINT(10 10)', 'POINT(20 0)'));

degrees

 270
```

Angle between vectors defined by four points

```
SELECT degrees(ST_Angle('POINT (10 10)', 'POINT (0 0)', 'POINT(90 90)', 'POINT (100 80)')) ←
);

degrees

269.99999999999999
```

Angle between vectors defined by the start and end points of lines

```
SELECT degrees(ST_Angle('LINESTRING(0 0, 0.3 0.7, 1 1)', 'LINESTRING(0 0, 0.2 0.5, 1 0)')) ←
);

degrees

 45
```

Ver también

[ST\\_Azimuth](#)

### 7.12.4 ST\_ClosestPoint

**ST\_ClosestPoint** — Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.

#### Synopsis

geometry **ST\_ClosestPoint**(geometry geom1, geometry geom2);  
geography **ST\_ClosestPoint**(geography geom1, geography geom2, boolean use\_spheroid = true);

#### Descripción

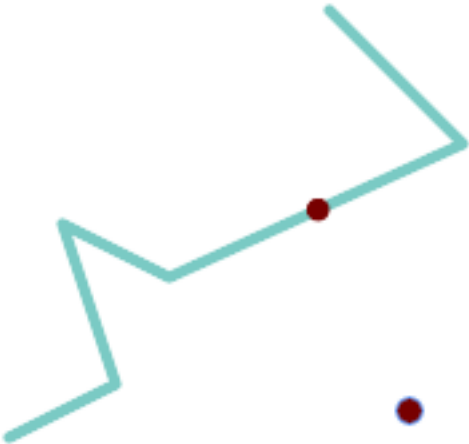
Returns the 2-dimensional point on geom1 that is closest to geom2. This is the first point of the shortest line between the geometries (as computed by **ST\_ShortestLine**).



**Note**  
If you have a 3D Geometry, you may prefer to use **ST\_3DClosestPoint**.

Enhanced: 3.4.0 - Support for geography.  
Disponibilidad: 1.5.0

#### Ejemplos

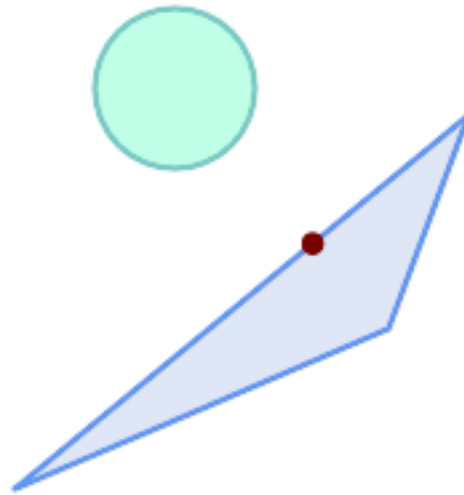


*The closest point for a Point and a LineString is the point itself. The closest point for a LineString and a Point is a point on the line.*

```
SELECT ST_AsText(ST_ClosestPoint(pt,line)) AS cp_pt_line,
 ST_AsText(ST_ClosestPoint(line,pt)) AS cp_line_pt
FROM (SELECT 'POINT (160 40)::geometry AS pt,
 'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)::geometry AS line) AS t;
```

| cp_pt_line    | cp_line_pt                                   |
|---------------|----------------------------------------------|
| POINT(160 40) | POINT(125.75342465753425 115.34246575342466) |





*The closest point on polygon A to polygon B*

```
SELECT ST_AsText(ST_ClosestPoint(
 'POLYGON ((190 150, 20 10, 160 70, 190 150))',
 ST_Buffer('POINT(80 160)', 30)
)) As ptwkt;

POINT(131.59149149528952 101.89887534906197)
```

#### Ver también

[ST\\_3DClosestPoint](#), [ST\\_Distance](#), [ST\\_LongestLine](#), [ST\\_ShortestLine](#), [ST\\_MaxDistance](#)

### 7.12.5 ST\_3DClosestPoint

**ST\_3DClosestPoint** — Returns the 3D point on g1 that is closest to g2. This is the first point of the 3D shortest line.

#### Synopsis

geometry **ST\_3DClosestPoint**(geometry g1, geometry g2);

#### Descripción

Returns the 3-dimensional point on g1 that is closest to g2. This is the first point of the 3D shortest line. The 3D length of the 3D shortest line is the 3D distance.



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



This function supports Polyhedral surfaces.

Disponibilidad: 2.0.0

Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.

#### Ejemplos

linestring and point -- both 3d and 2d closest point

```
SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,
 ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt
FROM (SELECT 'POINT(100 100 30)::geometry As pt,
 'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)::' AS
 geometry As line
) As foo;
```

| cp3d_line_pt                                              |  |                                          |
|-----------------------------------------------------------|--|------------------------------------------|
| POINT(54.6993798867619 128.935022917228 11.5475869506606) |  | POINT(73.0769230769231 115.384615384615) |

linestring and multipoint -- both 3d and 2d closest point

```
SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,
 ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)::geometry As pt,
 'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)::' AS
 geometry As line
) As foo;
```

| cp3d_line_pt                                              |  | cp2d_line_pt |
|-----------------------------------------------------------|--|--------------|
| POINT(54.6993798867619 128.935022917228 11.5475869506606) |  | POINT(50 75) |

Multilinestring and polygon both 3d and 2d closest point

```
SELECT ST_AsEWKT(ST_3DClosestPoint(poly, mline)) As cp3d,
 ST_AsEWKT(ST_ClosestPoint(poly, mline)) As cp2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,
100 100 5, 175 150 5))') As poly,
 ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125
100 1, 175 155 1),
(1 10 2, 5 20 1))') As mline) As foo;
```

| cp3d                                      |  | cp2d         |
|-------------------------------------------|--|--------------|
| POINT(39.993580415989 54.1889925532825 5) |  | POINT(20 40) |

Ver también

[ST\\_AsEWKT](#), [ST\\_ClosestPoint](#), [ST\\_3DDistance](#), [ST\\_3DShortestLine](#)

7.12.6 ST\_Distance

ST\_Distance — Returns the distance between two geometry or geography values.

Synopsis

float **ST\_Distance**(geometry g1, geometry g2);  
float **ST\_Distance**(geography geog1, geography geog2, boolean use\_spheroid = true);

## Descripción

For **geometry** types returns the minimum 2D Cartesian (planar) distance between two geometries, in projected units (spatial ref units).

For **geography** types defaults to return the minimum geodesic distance between two geographies in meters, compute on the spheroid determined by the SRID. If `use_spheroid` is false, a faster spherical calculation is used.



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



This method implements the SQL/MM specification. SQL-MM 3: 5.1.23



This method supports Circular Strings and Curves.

Availability: 1.5.0 geography support was introduced in 1.5. Speed improvements for planar to better handle large or many vertex geometries

Enhanced: 2.1.0 improved speed for geography. See **Making Geography faster** for details.

Enhanced: 2.1.0 - support for curved geometries was introduced.

Enhanced: 2.2.0 - measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires PROJ >= 4.9.0 to take advantage of the new feature.

Changed: 3.0.0 - does not depend on SFCGAL anymore.

## Geometry Examples

Geometry example - units in planar degrees 4326 is WGS 84 long lat, units are degrees.

```
SELECT ST_Distance(
 'SRID=4326;POINT(-72.1235 42.3521)::geometry',
 'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry ');

0.00150567726382282
```

Geometry example - units in meters (SRID: 3857, proportional to pixels on popular web maps). Although the value is off, nearby ones can be compared correctly, which makes it a good choice for algorithms like KNN or KMeans.

```
SELECT ST_Distance(
 ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry', 3857),
 ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry', 3857)) ←
 ;

167.441410065196
```

Geometry example - units in meters (SRID: 3857 as above, but corrected by cos(lat) to account for distortion)

```
SELECT ST_Distance(
 ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry', 3857),
 ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry', 3857)
) * cosd(42.3521);

123.742351254151
```

Geometry example - units in meters (SRID: 26986 Massachusetts state plane meters) (most accurate for Massachusetts)

```
SELECT ST_Distance(
 ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry', 26986),
 ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry', 26986) ←
);

123.797937878454
```

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (least accurate)

```
SELECT ST_Distance(
 ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry, 2163),
 ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry, 2163)) ←
 ;

126.664256056812
```

Geography Examples

Same as geometry example but note units in meters - use sphere for slightly faster and less accurate computation.

```
SELECT ST_Distance(gg1, gg2) As spheroid_dist, ST_Distance(gg1, gg2, false) As sphere_dist
FROM (SELECT
 'SRID=4326;POINT(-72.1235 42.3521)::geography as gg1,
 'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geography as gg2
) As foo ;

spheroid_dist | sphere_dist
-----+-----
123.802076746848 | 123.475736916397
```

Ver también

[ST\\_3DDistance](#), [ST\\_DWithin](#), [ST\\_DistanceSphere](#), [ST\\_DistanceSpheroid](#), [ST\\_MaxDistance](#), [ST\\_HausdorffDistance](#), [ST\\_FrechetDistance](#), [ST\\_Transform](#)

7.12.7 ST\_3DDistance

ST\_3DDistance — Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units.

Synopsis

float **ST\_3DDistance**(geometry g1, geometry g2);

Descripción

Returns the 3-dimensional minimum cartesian distance between two geometries in projected units (spatial ref units).

-  This function supports 3d and will not drop the z-index.
-  This function supports Polyhedral surfaces.
-  This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3

Disponibilidad: 2.0.0

Changed: 2.2.0 - In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.

Changed: 3.0.0 - SFCGAL version removed

Ejemplos

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point ←
and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ←
units as final.
SELECT ST_3DDistance(
 ST_Transform('SRID=4326;POINT(-72.1235 42.3521 4) '::geometry,2163),
 ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 20) '::geometry,2163)
) As dist_3d,
ST_Distance(
 ST_Transform('SRID=4326;POINT(-72.1235 42.3521) '::geometry,2163),
 ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546) '::geometry,2163)
) As dist_2d;

dist_3d | dist_2d
-----+-----
127.295059324629 | 126.66425605671

-- Multilinestring and polygon both 3d and 2d distance
-- Same example as 3D closest point example
SELECT ST_3DDistance(poly, mline) As dist3d,
 ST_Distance(poly, mline) As dist2d
 FROM (SELECT 'POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, 100 100 5, 175 150 5) ←
) '::geometry as poly,
 'MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 100 1, 175 155 1), (1 ←
10 2, 5 20 1))'::geometry as mline) as foo;

dist3d | dist2d
-----+-----
0.716635696066337 | 0
```

Ver también

[ST\\_Distance](#), [ST\\_3DClosestPoint](#), [ST\\_3DDWithin](#), [ST\\_3DMaxDistance](#), [ST\\_3DShortestLine](#), [ST\\_Transform](#)

7.12.8 ST\_DistanceSphere

ST\_DistanceSphere — Returns minimum distance in meters between two lon/lat geometries using a spherical earth model.

Synopsis

float **ST\_DistanceSphere**(geometry geomlonlatA, geometry geomlonlatB, float8 radius=6371008);

Descripción

Returns minimum distance in meters between two lon/lat points. Uses a spherical earth and radius derived from the spheroid defined by the SRID. Faster than [ST\\_DistanceSpheroid](#), but less accurate. PostGIS Versions prior to 1.5 only implemented for points.

Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points.

Changed: 2.2.0 In prior versions this used to be called ST\_Distance\_Sphere

Ejemplos

```
SELECT round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38)'
 ',4326)) As numeric),2) As dist_meters,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(geom),32611),
 ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) As
 As dist_utm11_meters,
round(CAST(ST_Distance(geom), ST_GeomFromText('POINT(-118 38)', 4326)) As
 numeric),5) As dist_degrees,
round(CAST(ST_Distance(ST_Transform(geom,32611),
 ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) As
 As min_dist_line_point_meters
FROM
 (SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As geom) As
 as foo;
dist_meters | dist_utm11_meters | dist_degrees | min_dist_line_point_meters
-----+-----+-----+-----
70424.47 | 70438.00 | 0.72900 | 65871.18
```

Ver también

[ST\\_Distance](#), [ST\\_DistanceSpheroid](#)

7.12.9 ST\_DistanceSpheroid

ST\_DistanceSpheroid — Returns the minimum distance between two lon/lat geometries using a spheroidal earth model.

Synopsis

float **ST\_DistanceSpheroid**(geometry geomlonlatA, geometry geomlonlatB, spheroid measurement\_spheroid=WGS84);

Descripción

Returns minimum distance in meters between two lon/lat geometries given a particular spheroid. See the explanation of spheroids given for [ST\\_LengthSpheroid](#).



**Note**  
This function does not look at the SRID of the geometry. It assumes the geometry coordinates are based on the provided spheroid.

Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points.  
Changed: 2.2.0 In prior versions this was called ST\_Distance\_Spheroid

Ejemplos

```
SELECT round(CAST(
 ST_DistanceSpheroid(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38)'
 ',4326), 'SPHEROID["WGS 84",6378137,298.257223563]')
 As numeric),2) As dist_meters_spheroid,
round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118
 38)',4326)) As numeric),2) As dist_meters_sphere,
```

```
round(CAST(ST_Distance(ST_Transform(ST_Centroid(geom), 32611),
 ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326), 32611)) As numeric), 2) ←
 As dist_utm11_meters
FROM
 (SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As geom) ←
 as foo;
dist_meters_spheroid | dist_meters_sphere | dist_utm11_meters
-----+-----+-----
70454.92 | 70424.47 | 70438.00
```

Ver también

[ST\\_Distance](#), [ST\\_DistanceSphere](#)

### 7.12.10 ST\_FrechetDistance

ST\_FrechetDistance — Returns the Fréchet distance between two geometries.

#### Synopsis

float **ST\_FrechetDistance**(geometry g1, geometry g2, float densifyFrac = -1);

#### Descripción

Implements algorithm for computing the Fréchet distance restricted to discrete points for both geometries, based on [Computing Discrete Fréchet Distance](#). The Fréchet distance is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. Therefore it is often better than the Hausdorff distance.

When the optional densifyFrac is specified, this function performs a segment densification before computing the discrete Fréchet distance. The densifyFrac parameter sets the fraction by which to densify each segment. Each segment will be split into a number of equal-length subsegments, whose fraction of the total length is closest to the given fraction.

Units are in the units of the spatial reference system of the geometries.



**Note**  
The current implementation supports only vertices as the discrete locations. This could be extended to allow an arbitrary density of points to be used.



**Note**  
The smaller densifyFrac we specify, the more accurate Fréchet distance we get. But, the computation time and the memory usage increase with the square of the number of subsegments.

Realizado por el módulo de GEOS  
Availability: 2.4.0 - requires GEOS >= 3.7.0

## Ejemplos

```
postgres=# SELECT st_frechetdistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, ↵
 50 50, 100 0)::geometry');
 st_frechetdistance

 70.7106781186548
(1 row)
```

```
SELECT st_frechetdistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, 50 50, 100 ↵
 0)::geometry, 0.5);
 st_frechetdistance

 50
(1 row)
```

## Ver también

[ST\\_HausdorffDistance](#)

### 7.12.11 ST\_HausdorffDistance

**ST\_HausdorffDistance** — Returns the Hausdorff distance between two geometries.

#### Synopsis

```
float ST_HausdorffDistance(geometry g1, geometry g2);
float ST_HausdorffDistance(geometry g1, geometry g2, float densifyFrac);
```

#### Descripción

Returns the **Hausdorff distance** between two geometries. The Hausdorff distance is a measure of how similar or dissimilar 2 geometries are.

The function actually computes the "Discrete Hausdorff Distance". This is the Hausdorff distance computed at discrete points on the geometries. The *densifyFrac* parameter can be specified, to provide a more accurate answer by densifying segments before computing the discrete Hausdorff distance. Each segment is split into a number of equal-length subsegments whose fraction of the segment length is closest to the given fraction.

Units are in the units of the spatial reference system of the geometries.



#### Note

This algorithm is NOT equivalent to the standard Hausdorff distance. However, it computes an approximation that is correct for a large subset of useful cases. One important case is Linestrings that are roughly parallel to each other, and roughly equal in length. This is a useful metric for line matching.

Disponibilidad: 1.5.0



## Ejemplos



*Hausdorff distance (red) and distance (yellow) between two lines*

```
SELECT ST_HausdorffDistance(geomA, geomB),
 ST_Distance(geomA, geomB)
FROM (SELECT 'LINESTRING (20 70, 70 60, 110 70, 170 70)::geometry AS geomA,
 'LINESTRING (20 90, 130 90, 60 100, 190 100)::geometry AS geomB) AS t;
st_hausdorffdistance | st_distance
-----+-----
37.26206567625497 | 20
```

### Example: Hausdorff distance with densification.

```
SELECT ST_HausdorffDistance(
 'LINESTRING (130 0, 0 0, 0 150)::geometry',
 'LINESTRING (10 10, 10 150, 130 10)::geometry',
 0.5);

70
```

**Example:** For each building, find the parcel that best represents it. First we require that the parcel intersect with the building geometry. `DISTINCT ON` guarantees we get each building listed only once. `ORDER BY .. ST_HausdorffDistance` selects the parcel that is most similar to the building.

```
SELECT DISTINCT ON (buildings.gid) buildings.gid, parcels.parcel_id
FROM buildings
 INNER JOIN parcels
 ON ST_Intersects(buildings.geom, parcels.geom)
ORDER BY buildings.gid, ST_HausdorffDistance(buildings.geom, parcels.geom);
```

## Ver también

[ST\\_FrechetDistance](#)

## 7.12.12 ST\_Length

`ST_Length` — Returns the 2D length of a linear geometry.

## Synopsis

```
float ST_Length(geometry a_2dlinestring);
float ST_Length(geography geog, boolean use_spheroid = true);
```

## Descripción

For geometry types: returns the 2D Cartesian length of the geometry if it is a LineString, MultiLineString, ST\_Curve, ST\_MultiCurve. For areal geometries 0 is returned; use **ST\_Perimeter** instead. The units of length is determined by the spatial reference system of the geometry.

For geography types: computation is performed using the inverse geodesic calculation. Units of length are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If `use_spheroid = false`, then the calculation is based on a sphere instead of a spheroid.

Currently for geometry this is an alias for ST\_Length2D, but this may change to support higher dimensions.



### Warning

Changed: 2.0.0 Breaking change -- in prior versions applying this to a MULTI/POLYGON of type geography would give you the perimeter of the POLYGON/MULTIPOLYGON. In 2.0.0 this was changed to return 0 to be in line with geometry behavior. Please use ST\_Perimeter if you want the perimeter of a polygon



### Note

For geography the calculation defaults to using a spheroidal model. To use the faster but less accurate spherical calculation use ST\_Length(gg,false);



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.5.1



This method implements the SQL/MM specification. SQL-MM 3: 7.1.2, 9.3.4

Availability: 1.5.0 geography support was introduced in 1.5.

## Geometry Examples

Return length in feet for line string. Note this is in feet because EPSG:2249 is Massachusetts State Plane Feet

```
SELECT ST_Length(ST_GeomFromText('LINESTRING(743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416)',2249));

st_length

122.630744000095

--Transforming WGS 84 LineString to Massachusetts state plane meters
SELECT ST_Length(
 ST_Transform(
 ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, ↵
 -72.123 42.1546)'),
 26986
)
);

st_length

34309.4563576191
```

## Geography Examples

Return length of WGS 84 geography line

```
-- the default calculation uses a spheroid
SELECT ST_Length(the_geog) As length_spheroid, ST_Length(the_geog,false) As length_sphere
FROM (SELECT ST_GeographyFromText(
'SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, -72.123 42.1546)') As the_geog)
As foo;
```

| length_spheroid  |  | length_sphere    |
|------------------|--|------------------|
| 34310.5703627288 |  | 34346.2060960742 |

Ver también

[ST\\_GeographyFromText](#), [ST\\_GeomFromEWKT](#), [ST\\_LengthSpheroid](#), [ST\\_Perimeter](#), [ST\\_Transform](#)

### 7.12.13 ST\_Length2D

**ST\_Length2D** — Returns the 2D length of a linear geometry. Alias for `ST_Length`

#### Synopsis

```
float ST_Length2D(geometry a_2dlinestring);
```

#### Descripción

Returns the 2D length of the geometry if it is a linestring or multi-linestring. This is an alias for `ST_Length`

Ver también

[ST\\_Length](#), [ST\\_3DLength](#)

### 7.12.14 ST\_3DLength

**ST\_3DLength** — Returns the 3D length of a linear geometry.

#### Synopsis

```
float ST_3DLength(geometry a_3dlinestring);
```

#### Descripción

Returns the 3-dimensional or 2-dimensional length of the geometry if it is a `LineString` or `MultiLineString`. For 2-d lines it will just return the 2-d length (same as `ST_Length` and `ST_Length2D`)



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



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 7.1, 10.3

Changed: 2.0.0 In prior versions this used to be called `ST_Length3D`

## Ejemplos

Return length in feet for a 3D cable. Note this is in feet because EPSG:2249 is Massachusetts State Plane Feet

```
SELECT ST_3DLength(ST_GeomFromText('LINESTRING(743238 2967416 1,743238 2967450 1,743265 2967450 3,743265.625 2967416 3,743238 2967416 3)',2249));
ST_3DLength

122.704716741457
```

## Ver también

[ST\\_Length](#), [ST\\_Length2D](#)

### 7.12.15 ST\_LengthSpheroid

**ST\_LengthSpheroid** — Returns the 2D or 3D length/perimeter of a lon/lat geometry on a spheroid.

## Synopsis

float **ST\_LengthSpheroid**(geometry a\_geometry, spheroid a\_spheroid);

## Descripción

Calculates the length or perimeter of a geometry on an ellipsoid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection. The spheroid is specified by a text value as follows:

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

## Ejemplos

```
SPHEROID["GRS_1980",6378137,298.257222101]
```

Disponibilidad: 1.2.2

Changed: 2.2.0 In prior versions this was called ST\_Length\_Spheroid and had the alias ST\_3DLength\_Spheroid



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

## Ejemplos

```
SELECT ST_LengthSpheroid(geometry_column,
 'SPHEROID["GRS_1980",6378137,298.257222101]')
FROM geometry_table;

SELECT ST_LengthSpheroid(geom, sph_m) As tot_len,
ST_LengthSpheroid(ST_GeometryN(geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(geom,2), sph_m) As len_line2
FROM (SELECT ST_GeomFromText('MULTILINESTRING((-118.584 38.374,-118.583 38.5),
(-71.05957 42.3589 , -71.061 43))') As geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m) as foo;
tot_len | len_line1 | len_line2
-----+-----+-----
85204.5207562955 | 13986.8725229309 | 71217.6482333646
```

```
--3D
SELECT ST_LengthSpheroid(geom, sph_m) As tot_len,
ST_LengthSpheroid(ST_GeometryN(geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(geom,2), sph_m) As len_line2
 FROM (SELECT ST_GeomFromEWKT('MULTILINESTRING((-118.584 38.374 ↵
 20,-118.583 38.5 30),
 (-71.05957 42.3589 75, -71.061 43 90))') As geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m) as foo;

 tot_len | len_line1 | len_line2
-----+-----+-----
85204.5259107402 | 13986.876097711 | 71217.6498130292
```

Ver también

[ST\\_GeometryN](#), [ST\\_Length](#)

7.12.16 ST\_LongestLine

ST\_LongestLine — Returns the 2D longest line between two geometries.

Synopsis

geometry **ST\_LongestLine**(geometry g1, geometry g2);

Descripción

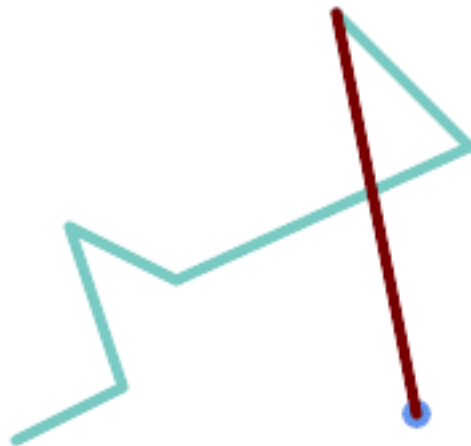
Returns the 2-dimensional longest line between the points of two geometries. The line returned starts on g1 and ends on g2.

The longest line always occurs between two vertices. The function returns the first longest line if more than one is found. The length of the line is equal to the distance returned by [ST\\_MaxDistance](#).

If g1 and g2 are the same geometry, returns the line between the two vertices farthest apart in the geometry. The endpoints of the line lie on the circle computed by [ST\\_MinimumBoundingCircle](#).

Disponibilidad: 1.5.0

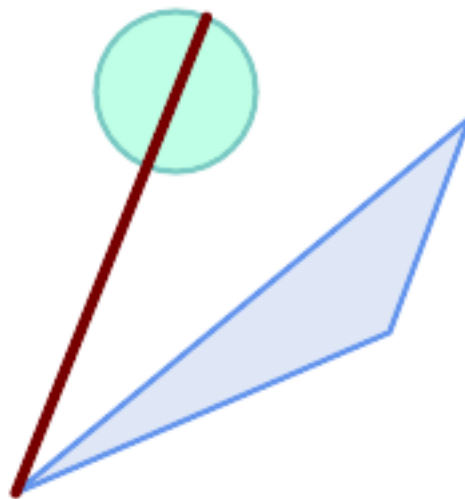
## Ejemplos



*Longest line between a point and a line*

```
SELECT ST_AsText(ST_LongestLine(
 'POINT (160 40)',
 'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)')
) AS lline;

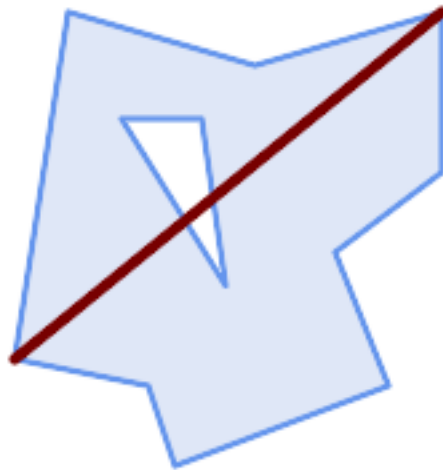
LINESTRING(160 40,130 190)
```



*Longest line between two polygons*

```
SELECT ST_AsText(ST_LongestLine(
 'POLYGON ((190 150, 20 10, 160 70, 190 150))',
 ST_Buffer('POINT(80 160)', 30)
)) AS lline;

LINESTRING(20 10,105.3073372946034 186.95518130045156)
```



Longest line across a single geometry. The length of the line is equal to the Maximum Distance. The endpoints of the line lie on the Minimum Bounding Circle.

```
SELECT ST_AsText(ST_LongestLine(geom, geom)) AS llinewkt,
 ST_MaxDistance(geom, geom) AS max_dist,
 ST_Length(ST_LongestLine(geom, geom)) AS lenll
FROM (SELECT 'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 50, 40 180),
 (60 140, 99 77.5, 90 140, 60 140))'::geometry AS geom) AS t;
```

| llinewkt                  | max_dist           | lenll              |
|---------------------------|--------------------|--------------------|
| LINESTRING(20 50,180 180) | 206.15528128088303 | 206.15528128088303 |

Ver también

[ST\\_MaxDistance](#), [ST\\_ShortestLine](#), [ST\\_3DLongestLine](#), [ST\\_MinimumBoundingCircle](#)

7.12.17 ST\_3DLongestLine

ST\_3DLongestLine — Returns the 3D longest line between two geometries

Synopsis

geometry **ST\_3DLongestLine**(geometry g1, geometry g2);

Descripción

Returns the 3-dimensional longest line between two geometries. The function returns the first longest line if more than one. The line returned starts in g1 and ends in g2. The 3D length of the line is equal to the distance returned by [ST\\_3DMaxDistance](#).

Disponibilidad: 2.0.0

Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.

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

Ejemplos

linestring and point -- both 3d and 2d longest line

```
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
 ST_AsEWKT(ST_LongestLine(line,pt)) As lol2d_line_pt
FROM (SELECT 'POINT(100 100 30)::geometry As pt,
 'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)::' ␣
 geometry As line
) As foo;
```

| lol3d_line_pt                                                  |  | lol2d_line_pt |
|----------------------------------------------------------------|--|---------------|
| -----+-----                                                    |  |               |
| LINESTRING(50 75 1000,100 100 30)   LINESTRING(98 190,100 100) |  |               |

linestring and multipoint -- both 3d and 2d longest line

```
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
 ST_AsEWKT(ST_LongestLine(line,pt)) As lol2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)::geometry As pt,
 'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)::' ␣
 geometry As line
) As foo;
```

| lol3d_line_pt                                              |  | lol2d_line_pt |
|------------------------------------------------------------|--|---------------|
| -----+-----                                                |  |               |
| LINESTRING(98 190 1,50 74 1000)   LINESTRING(98 190,50 74) |  |               |

MultiLineString and Polygon both 3d and 2d longest line

```
SELECT ST_AsEWKT(ST_3DLongestLine(poly, mline)) As lol3d,
 ST_AsEWKT(ST_LongestLine(poly, mline)) As lol2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, ␣
100 100 5, 175 150 5))') As poly,
 ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 ␣
100 1, 175 155 1),
 (1 10 2, 5 20 1))') As mline) As foo;
```

| lol3d                                                   |  | lol2d |
|---------------------------------------------------------|--|-------|
| -----+-----                                             |  |       |
| LINESTRING(175 150 5,1 10 2)   LINESTRING(175 150,1 10) |  |       |

Ver también

[ST\\_3DClosestPoint](#), [ST\\_3DDistance](#), [ST\\_LongestLine](#), [ST\\_3DShortestLine](#), [ST\\_3DMaxDistance](#)

7.12.18 ST\_MaxDistance

ST\_MaxDistance — Returns the 2D largest distance between two geometries in projected units.

Synopsis

float **ST\_MaxDistance**(geometry g1, geometry g2);



**Descripción**

Returns the 2-dimensional maximum distance between two geometries, in projected units. The maximum distance always occurs between two vertices. This is the length of the line returned by [ST\\_LongestLine](#).

If g1 and g2 are the same geometry, returns the distance between the two vertices farthest apart in that geometry.

Disponibilidad: 1.5.0

**Ejemplos**

Maximum distance between a point and lines.

```
SELECT ST_MaxDistance('POINT(0 0)::geometry', 'LINESTRING (2 0, 0 2) '::geometry);

2

SELECT ST_MaxDistance('POINT(0 0)::geometry', 'LINESTRING (2 2, 2 2) '::geometry);

2.82842712474619
```

Maximum distance between vertices of a single geometry.

```
SELECT ST_MaxDistance('POLYGON ((10 10, 10 0, 0 0, 10 10)) '::geometry,
 'POLYGON ((10 10, 10 0, 0 0, 10 10)) '::geometry);

14.142135623730951
```

**Ver también**

[ST\\_Distance](#), [ST\\_LongestLine](#), [ST\\_DFullyWithin](#)

**7.12.19 ST\_3DMaxDistance**

**ST\_3DMaxDistance** — Returns the 3D cartesian maximum distance (based on spatial ref) between two geometries in projected units.

**Synopsis**

```
float ST_3DMaxDistance(geometry g1, geometry g2);
```

**Descripción**

Returns the 3-dimensional maximum cartesian distance between two geometries in projected units (spatial ref units).



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



This function supports Polyhedral surfaces.

Disponibilidad: 2.0.0

Changed: 2.2.0 - In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.

## Ejemplos

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point ↔
 and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ↔
 units as final.
SELECT ST_3DMaxDistance(
 ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 10000)'),2163),
 ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 20)'),2163)
) As dist_3d,
ST_MaxDistance(
 ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 10000)'),2163),
 ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 20)'),2163)
) As dist_2d;

dist_3d | dist_2d
-----+-----
24383.7467488441 | 22247.8472107251
```

## Ver también

[ST\\_Distance](#), [ST\\_3DDWithin](#), [ST\\_3DMaxDistance](#), [ST\\_Transform](#)

### 7.12.20 ST\_MinimumClearance

**ST\_MinimumClearance** — Returns the minimum clearance of a geometry, a measure of a geometry's robustness.

#### Synopsis

```
float ST_MinimumClearance(geometry g);
```

#### Descripción

It is possible for a geometry to meet the criteria for validity according to [ST\\_IsValid](#) (polygons) or [ST\\_IsSimple](#) (lines), but to become invalid if one of its vertices is moved by a small distance. This can happen due to loss of precision during conversion to text formats (such as WKT, KML, GML, GeoJSON), or binary formats that do not use double-precision floating point coordinates (e.g. MapInfo TAB).

The minimum clearance is a quantitative measure of a geometry's robustness to change in coordinate precision. It is the largest distance by which vertices of the geometry can be moved without creating an invalid geometry. Larger values of minimum clearance indicate greater robustness.

If a geometry has a minimum clearance of  $\epsilon$ , then:

- No two distinct vertices in the geometry are closer than the distance  $\epsilon$ .
- No vertex is closer than  $\epsilon$  to a line segment of which it is not an endpoint.

If no minimum clearance exists for a geometry (e.g. a single point, or a MultiPoint whose points are identical), the return value is *Infinity*.

To avoid validity issues caused by precision loss, [ST\\_ReducePrecision](#) can reduce coordinate precision while ensuring that polygonal geometry remains valid.

Disponibilidad: 2.3.0

## Ejemplos

```
SELECT ST_MinimumClearance('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))');
st_minimumclearance

0.00032
```

## Ver también

[ST\\_MinimumClearanceLine](#), [ST\\_Distance](#), [ST\\_LongestLine](#), [ST\\_MaxDistance](#)

### 7.12.21 ST\_MinimumClearanceLine

**ST\_MinimumClearanceLine** — Returns the two-point LineString spanning a geometry's minimum clearance.

## Synopsis

Geometry **ST\_MinimumClearanceLine**(geometry g);

## Descripción

Returns the two-point LineString spanning a geometry's minimum clearance. If the geometry does not have a minimum clearance, `LINESTRING EMPTY` is returned.

Realizado por el módulo de GEOS

Availability: 2.3.0 - requires GEOS >= 3.6.0

## Ejemplos

```
SELECT ST_AsText(ST_MinimumClearanceLine('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))'));

LINESTRING(0.5 0.00032,0.5 0)
```

## Ver también

[ST\\_MinimumClearance](#)

### 7.12.22 ST\_Perimeter

**ST\_Perimeter** — Returns the length of the boundary of a polygonal geometry or geography.

## Synopsis

```
float ST_Perimeter(geometry g1);
float ST_Perimeter(geography geog, boolean use_spheroid = true);
```

## Descripción

Returns the 2D perimeter of the geometry/geography if it is a ST\_Surface, ST\_MultiSurface (Polygon, MultiPolygon). 0 is returned for non-areal geometries. For linear geometries use [ST\\_Length](#). For geometry types, units for perimeter measures are specified by the spatial reference system of the geometry.

For geography types, the calculations are performed using the inverse geodesic problem, where perimeter units are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If `use_spheroid = false`, then calculations will approximate a sphere instead of a spheroid.

Currently this is an alias for ST\_Perimeter2D, but this may change to support higher dimensions.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.5.1



This method implements the SQL/MM specification. SQL-MM 3: 8.1.3, 9.5.4

Availability 2.0.0: Support for geography was introduced

## Ejemplos: Geometry

Return perimeter in feet for Polygon and MultiPolygon. Note this is in feet because EPSG:2249 is Massachusetts State Plane Feet

```
SELECT ST_Perimeter(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416))', 2249));
st_perimeter

122.630744000095
(1 row)

SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((763104.471273676 2949418.44119003,
763104.477769673 2949418.42538203,
763104.189609677 2949418.22343004,763104.471273676 2949418.44119003)),
((763104.471273676 2949418.44119003,763095.804579742 2949436.33850239,
763086.132105649 2949451.46730207,763078.452329651 2949462.11549407,
763075.354136904 2949466.17407812,763064.362142565 2949477.64291974,
763059.953961626 2949481.28983009,762994.637609571 2949532.04103014,
762990.568508415 2949535.06640477,762986.710889563 2949539.61421415,
763117.237897679 2949709.50493431,763235.236617789 2949617.95619822,
763287.718121842 2949562.20592617,763111.553321674 2949423.91664605,
763104.471273676 2949418.44119003)))', 2249));
st_perimeter

845.227713366825
(1 row)
```

## Ejemplos: Geography

Return perimeter in meters and feet for Polygon and MultiPolygon. Note this is geography (WGS 84 long lat)

```
SELECT ST_Perimeter(geog) As per_meters, ST_Perimeter(geog)/0.3048 As per_ft
FROM ST_GeogFromText('POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 ↔
42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.1776848522251 ↔
42.3902896512902))') As geog;

per_meters | per_ft
-----+-----
37.3790462565251 | 122.634666195949
```

```
-- MultiPolygon example --
SELECT ST_Perimeter(geog) As per_meters, ST_Perimeter(geog,false) As per_sphere_meters,
 ST_Perimeter(geog)/0.3048 As per_ft
FROM ST_GeogFromText('MULTIPOLYGON((((-71.1044543107478 42.340674480411,-71.1044542869917
42.3406744369506,
-71.1044553562977 42.340673886454,-71.1044543107478 42.340674480411)),
((-71.1044543107478 42.340674480411,-71.1044860600303 42.3407237015564,-71.1045215770124
42.3407653385914,
-71.1045498002983 42.3407946553165,-71.1045611902745 42.3408058316308,-71.1046016507427
42.340837442371,
-71.104617893173 42.3408475056957,-71.1048586153981 42.3409875993595,-71.1048736143677
42.3409959528211,
-71.1048878050242 42.3410084812078,-71.1044020965803 42.3414730072048,
-71.1039672113619 42.3412202916693,-71.1037740497748 42.3410666421308,
-71.1044280218456 42.3406894151355,-71.1044543107478 42.340674480411)))') As geog;
```

| per_meters       | per_sphere_meters | per_ft           |
|------------------|-------------------|------------------|
| 257.634283683311 | 257.412311446337  | 845.256836231335 |

Ver también

[ST\\_GeogFromText](#), [ST\\_GeomFromText](#), [ST\\_Length](#)

7.12.23 ST\_Perimeter2D

ST\_Perimeter2D — Returns the 2D perimeter of a polygonal geometry. Alias for ST\_Perimeter.

Synopsis

float ST\_Perimeter2D(geometry geomA);

Descripción

Returns the 2-dimensional perimeter of a polygonal geometry.



**Note**  
This is currently an alias for ST\_Perimeter. In future versions ST\_Perimeter may return the highest dimension perimeter for a geometry. This is still under consideration

Ver también

[ST\\_Perimeter](#)

7.12.24 ST\_3DPerimeter

ST\_3DPerimeter — Returns the 3D perimeter of a polygonal geometry.

Synopsis

float ST\_3DPerimeter(geometry geomA);

## Descripción

Returns the 3-dimensional perimeter of the geometry, if it is a polygon or multi-polygon. If the geometry is 2-dimensional, then the 2-dimensional perimeter is returned.



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



This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.1, 10.5

Changed: 2.0.0 In prior versions this used to be called ST\_Perimeter3D

## Ejemplos

Perimeter of a slightly elevated polygon in the air in Massachusetts state plane feet

```
SELECT ST_3DPerimeter(geom), ST_Perimeter2d(geom), ST_Perimeter(geom) FROM
 (SELECT ST_GeomFromEWKT('SRID=2249;POLYGON((743238 2967416 2,743238 ↵
 2967450 1,
743265.625 2967416 1,743238 2967416 2))') As geom) As foo;
```

| ST_3DPerimeter   | st_perimeter2d   | st_perimeter     |
|------------------|------------------|------------------|
| 105.465793597674 | 105.432997272188 | 105.432997272188 |

## Ver también

[ST\\_GeomFromEWKT](#), [ST\\_Perimeter](#), [ST\\_Perimeter2D](#)

## 7.12.25 ST\_ShortestLine

ST\_ShortestLine — Returns the 2D shortest line between two geometries

### Synopsis

```
geometry ST_ShortestLine(geometry geom1, geometry geom2);
geography ST_ShortestLine(geography geom1, geography geom2, boolean use_spheroid = true);
```

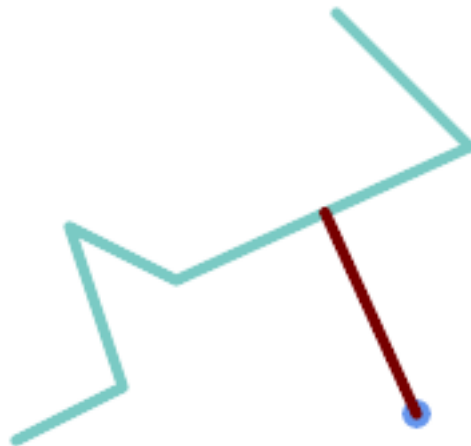
### Descripción

Returns the 2-dimensional shortest line between two geometries. The line returned starts in `geom1` and ends in `geom2`. If `geom1` and `geom2` intersect the result is a line with start and end at an intersection point. The length of the line is the same as [ST\\_Distance](#) returns for `g1` and `g2`.

Enhanced: 3.4.0 - support for geography.

Disponibilidad: 1.5.0

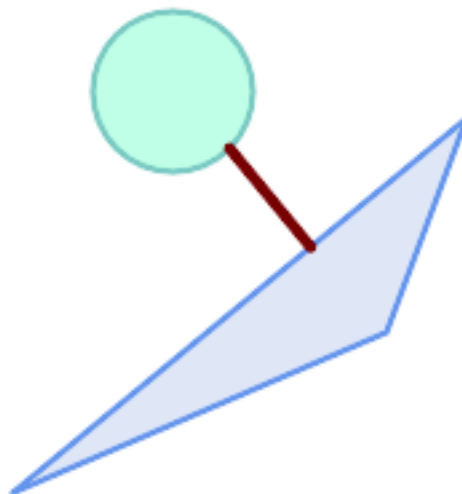
## Ejemplos



*Shortest line between Point and LineString*

```
SELECT ST_AsText(ST_ShortestLine(
 'POINT (160 40)',
 'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)')
) As sline;

LINESTRING(160 40,125.75342465753425 115.34246575342466)
```



*Shortest line between Polygons*

```
SELECT ST_AsText(ST_ShortestLine(
 'POLYGON ((190 150, 20 10, 160 70, 190 150))',
 ST_Buffer('POINT(80 160)', 30)
)) AS llinevkt;

LINESTRING(131.59149149528952 101.89887534906197,101.21320343559644 138.78679656440357)
```

Ver también

[ST\\_ClosestPoint](#), [ST\\_Distance](#), [ST\\_LongestLine](#), [ST\\_MaxDistance](#)

### 7.12.26 ST\_3DShortestLine

ST\_3DShortestLine — Returns the 3D shortest line between two geometries

#### Synopsis

geometry **ST\_3DShortestLine**(geometry g1, geometry g2);

#### Descripción

Returns the 3-dimensional shortest line between two geometries. The function will only return the first shortest line if more than one, that the function finds. If g1 and g2 intersects in just one point the function will return a line with both start and end in that intersection-point. If g1 and g2 are intersecting with more than one point the function will return a line with start and end in the same point but it can be any of the intersecting points. The line returned will always start in g1 and end in g2. The 3D length of the line this function returns will always be the same as [ST\\_3DDistance](#) returns for g1 and g2.

Disponibilidad: 2.0.0

Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.



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



This function supports Polyhedral surfaces.

#### Ejemplos

linestring and point -- both 3d and 2d shortest line

```
SELECT ST_AseWKT(ST_3DShortestLine(line,pt)) AS sh13d_line_pt,
 ST_AseWKT(ST_ShortestLine(line,pt)) As sh12d_line_pt
FROM (SELECT 'POINT(100 100 30)::geometry As pt,
 'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)::':: ↵
 geometry As line
) As foo;

sh13d_line_pt | ↵

sh12d_line_pt +-----+
LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30) | ↵
LINESTRING(73.0769230769231 115.384615384615,100 100)
```



|                                                                                                                                                                                                                                                                                                                                                                                               |                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| linestring and multipoint -- both 3d and 2d shortest line                                                                                                                                                                                                                                                                                                                                     |                   |
| <pre>SELECT ST_AsEWKT(ST_3DShortestLine(line,pt)) AS shl3d_line_pt,        ST_AsEWKT(ST_ShortestLine(line,pt)) As shl2d_line_pt FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000) '::geometry As pt,             'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900) ':: geometry As line       ) As foo;</pre>                                                                             |                   |
| shl2d_line_pt                                                                                                                                                                                                                                                                                                                                                                                 | shl3d_line_pt   ↵ |
| -----+-----                                                                                                                                                                                                                                                                                                                                                                                   |                   |
| LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30)   LINESTRING ↵<br>(50 75,50 74)                                                                                                                                                                                                                                                                                     |                   |
| MultiLineString and polygon both 3d and 2d shortest line                                                                                                                                                                                                                                                                                                                                      |                   |
| <pre>SELECT ST_AsEWKT(ST_3DShortestLine(poly, mline)) As shl3d,        ST_AsEWKT(ST_ShortestLine(poly, mline)) As shl2d FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, ↵ 100 100 5, 175 150 5))') As poly,        ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 ↵ 100 1, 175 155 1),        (1 10 2, 5 20 1))') As mline ) As foo;</pre> |                   |
| shl3d                                                                                                                                                                                                                                                                                                                                                                                         | shl2d   ↵         |
| -----+-----                                                                                                                                                                                                                                                                                                                                                                                   |                   |
| LINESTRING(39.993580415989 54.1889925532825 5,40.4078575708294 53.6052383805529 ↵<br>5.03423778139177)   LINESTRING(20 40,20 40)                                                                                                                                                                                                                                                              |                   |

Ver también

[ST\\_3DClosestPoint](#), [ST\\_3DDistance](#), [ST\\_LongestLine](#), [ST\\_ShortestLine](#), [ST\\_3DMaxDistance](#)

## 7.13 Overlay Functions

### 7.13.1 ST\_ClipByBox2D

ST\_ClipByBox2D — Computes the portion of a geometry falling within a rectangle.

#### Synopsis

geometry **ST\_ClipByBox2D**(geometry geom, box2d box);

#### Description

Clips a geometry by a 2D box in a fast and tolerant but possibly invalid way. Topologically invalid input geometries do not result in exceptions being thrown. The output geometry is not guaranteed to be valid (in particular, self-intersections for a polygon may be introduced).

Performed by the GEOS module.

Availability: 2.2.0

## Examples

```
-- Rely on implicit cast from geometry to box2d for the second parameter
SELECT ST_ClipByBox2D(geom, ST_MakeEnvelope(0,0,10,10)) FROM mytab;
```

## See Also

[ST\\_Intersection](#), [ST\\_MakeBox2D](#), [ST\\_MakeEnvelope](#)

## 7.13.2 ST\_Difference

**ST\_Difference** — Computes a geometry representing the part of geometry A that does not intersect geometry B.

### Synopsis

geometry **ST\_Difference**(geometry geomA, geometry geomB, float8 gridSize = -1);

### Description

Returns a geometry representing the part of geometry A that does not intersect geometry B. This is equivalent to  $A - \text{ST\_Intersection}(A, B)$ . If A is completely contained in B then an empty atomic geometry of appropriate type is returned.



#### Note

This is the only overlay function where input order matters. **ST\_Difference**(A, B) always returns a portion of A.

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

Performed by the GEOS module

Enhanced: 3.1.0 accept a `gridSize` parameter.

Requires GEOS  $\geq 3.9.0$  to use the `gridSize` parameter.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3

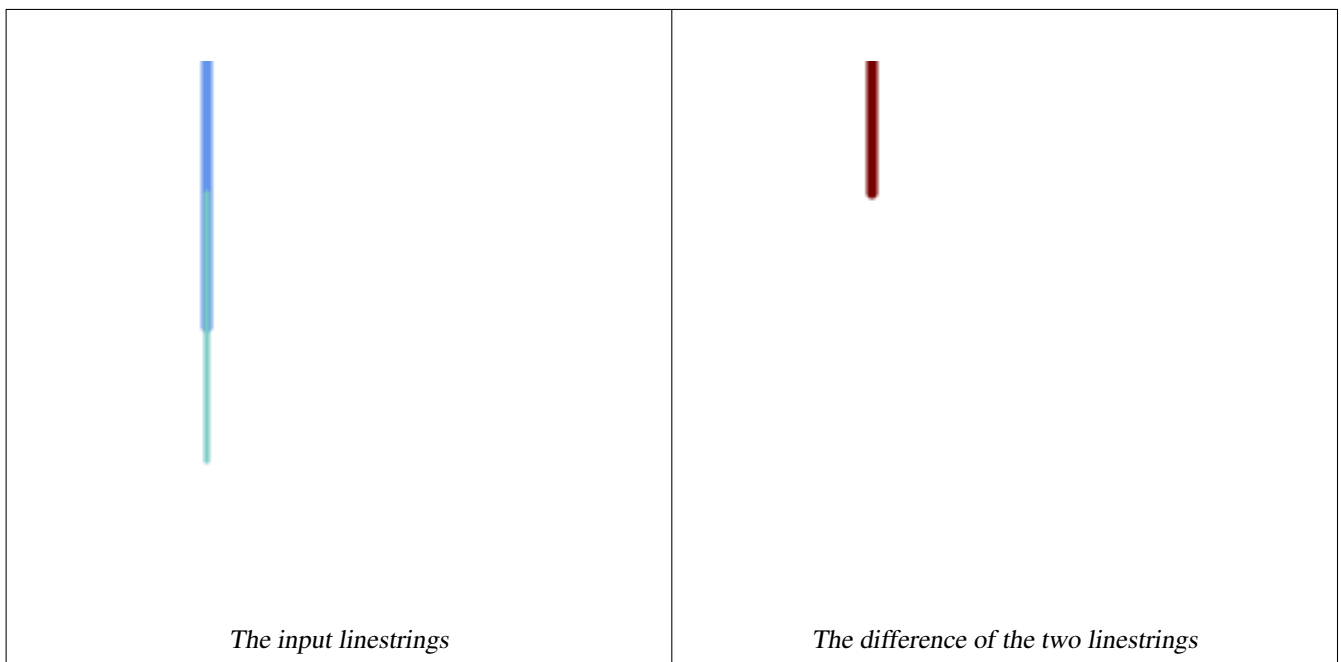


This method implements the SQL/MM specification. SQL-MM 3: 5.1.20



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

## Examples



The difference of 2D linestrings.

```
SELECT ST_AsText (
 ST_Difference (
 'LINESTRING(50 100, 50 200)'::geometry,
 'LINESTRING(50 50, 50 150)'::geometry
)
);

st_astext

LINESTRING(50 150,50 200)
```

The difference of 3D points.

```
SELECT ST_AsEWKT(ST_Difference(
 'MULTIPOINT(-118.58 38.38 5,-118.60 38.329 6,-118.614 38.281 7)' :: geometry,
 'POINT(-118.614 38.281 5)' :: geometry
));

st_asewkt

MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)
```

#### See Also

[ST\\_SymDifference](#), [ST\\_Intersection](#), [ST\\_Union](#)

### 7.13.3 ST\_Intersection

**ST\_Intersection** — Computes a geometry representing the shared portion of geometries A and B.

## Synopsis

```
geometry ST_Intersection(geometry geomA , geometry geomB , float8 gridSize = -1);
geography ST_Intersection(geography geogA , geography geogB);
```

## Description

Returns a geometry representing the point-set intersection of two geometries. In other words, that portion of geometry A and geometry B that is shared between the two geometries.

If the geometries have no points in common (i.e. are disjoint) then an empty atomic geometry of appropriate type is returned.

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

`ST_Intersection` in conjunction with `ST_Intersects` is useful for clipping geometries such as in bounding box, buffer, or region queries where you only require the portion of a geometry that is inside a country or region of interest.

### Note



For geography this is a thin wrapper around the geometry implementation. It first determines the best SRID that fits the bounding box of the 2 geography objects (if geography objects are within one half zone UTM but not same UTM will pick one of those) (favoring UTM or Lambert Azimuthal Equal Area (LAEA) north/south pole, and falling back on mercator in worst case scenario) and then intersection in that best fit planar spatial ref and retransforms back to WGS84 geography.



### Warning

This function will drop the M coordinate values if present.



### Warning

If working with 3D geometries, you may want to use SFGCAL based `ST_3DIntersection` which does a proper 3D intersection for 3D geometries. Although this function works with Z-coordinate, it does an averaging of Z-Coordinate.

Performed by the GEOS module

Enhanced: 3.1.0 accept a `gridSize` parameter

Requires GEOS  $\geq$  3.9.0 to use the `gridSize` parameter

Changed: 3.0.0 does not depend on SFCGAL.

Availability: 1.5 support for geography data type was introduced.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3



This method implements the SQL/MM specification. SQL-MM 3: 5.1.18



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

## Examples

```
SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING (2 0, 0 2)':: ↵
 geometry));
 st_astext

GEOMETRYCOLLECTION EMPTY

SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING (0 0, 0 2)':: ↵
 geometry));
 st_astext

POINT(0 0)
```

Clip all lines (trails) by country. Here we assume country geom are POLYGON or MULTIPOLYGONS. NOTE: we are only keeping intersections that result in a LINESTRING or MULTILINESTRING because we don't care about trails that just share a point. The dump is needed to expand a geometry collection into individual single MULT\* parts. The below is fairly generic and will work for polys, etc. by just changing the where clause.

```
select clipped.gid, clipped.f_name, clipped_geom
from (
 select trails.gid, trails.f_name,
 (ST_Dump(ST_Intersection(country.geom, trails.geom))).geom clipped_geom
 from country
 inner join trails on ST_Intersects(country.geom, trails.geom)
) as clipped
where ST_Dimension(clipped.clipped_geom) = 1;
```

For polys e.g. polygon landmarks, you can also use the sometimes faster hack that buffering anything by 0.0 except a polygon results in an empty geometry collection. (So a geometry collection containing polys, lines and points buffered by 0.0 would only leave the polygons and dissolve the collection shell.)

```
select poly.gid,
 ST_Multi(
 ST_Buffer(
 ST_Intersection(country.geom, poly.geom),
 0.0
)
) clipped_geom
from country
 inner join poly on ST_Intersects(country.geom, poly.geom)
where not ST_IsEmpty(ST_Buffer(ST_Intersection(country.geom, poly.geom), 0.0));
```

## Examples: 2.5Dish

Note this is not a true intersection, compare to the same example using [ST\\_3DIntersection](#).

```
select ST_AsText(ST_Intersection(linestring, polygon)) As wkt
from ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS ↵
 linestring
 CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

 st_astext

LINESTRING Z (1 1 8,0.5 0.5 8,0 0 10)
```

## See Also

[ST\\_3DIntersection](#), [ST\\_Difference](#), [ST\\_Union](#), [ST\\_Dimension](#), [ST\\_Dump](#), [ST\\_Force2D](#), [ST\\_SymDifference](#), [ST\\_Intersects](#), [ST\\_Multi](#)

### 7.13.4 ST\_MemUnion

ST\_MemUnion — Aggregate function which unions geometries in a memory-efficient but slower way

#### Synopsis

geometry **ST\_MemUnion**(geometry set geomfield);

#### Description

An aggregate function that unions the input geometries, merging them to produce a result geometry with no overlaps. The output may be a single geometry, a MultiGeometry, or a Geometry Collection.



#### Note

Produces the same result as **ST\_Union**, but uses less memory and more processor time. This aggregate function works by unioning the geometries incrementally, as opposed to the ST\_Union aggregate which first accumulates an array and then unions the contents using a fast algorithm.



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

#### Examples

```
SELECT id,
 ST_MemUnion(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

#### See Also

**ST\_Union**

### 7.13.5 ST\_Node

ST\_Node — Nodes a collection of lines.

#### Synopsis

geometry **ST\_Node**(geometry geom);

#### Description

Returns a (Multi)LineString representing the fully noded version of a collection of linestrings. The noding preserves all of the input nodes, and introduces the least possible number of new nodes. The resulting linework is dissolved (duplicate lines are removed).

This is a good way to create fully-noded linework suitable for use as input to **ST\_Polygonize**.

**ST\_UnaryUnion** can also be used to node and dissolve linework. It provides an option to specify a gridSize, which can provide simpler and more robust output. See also **ST\_Union** for an aggregate variant.



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

Performed by the GEOS module.

Availability: 2.0.0

Changed: 2.4.0 this function uses GEOSNode internally instead of GEOSUnaryUnion. This may cause the resulting linestrings to have a different order and direction compared to PostGIS < 2.4.

## Examples

Noding a 3D LineString which self-intersects

```
SELECT ST_AsText (
 ST_Node('LINESTRINGZ(0 0 0, 10 10 10, 0 10 5, 10 0 3)')::geometry
) As output;
output

MULTILINESTRING Z ((0 0 0,5 5 4.5),(5 5 4.5,10 10 10,0 10 5,5 5 4.5),(5 5 4.5,10 0 3))
```

Noding two LineStrings which share common linework. Note that the result linework is dissolved.

```
SELECT ST_AsText (
 ST_Node('MULTILINESTRING ((2 5, 2 1, 7 1), (6 1, 4 1, 2 3, 2 5))')::geometry
) As output;
output

MULTILINESTRING((2 5,2 3),(2 3,2 1,4 1),(4 1,2 3),(4 1,6 1),(6 1,7 1))
```

## See Also

[ST\\_UnaryUnion](#), [ST\\_Union](#)

## 7.13.6 ST\_Split

**ST\_Split** — Returns a collection of geometries created by splitting a geometry by another geometry.

### Synopsis

geometry **ST\_Split**(geometry input, geometry blade);

### Description

The function supports splitting a LineString by a (Multi)Point, (Multi)LineString or (Multi)Polygon boundary, or a (Multi)Polygon by a LineString. When a (Multi)Polygon is used as the blade, its linear components (the boundary) are used for splitting the input. The result geometry is always a collection.

This function is in a sense the opposite of [ST\\_Union](#). Applying ST\_Union to the returned collection should theoretically yield the original geometry (although due to numerical rounding this may not be exactly the case).



#### Note

If the the input and blade do not intersect due to numerical precision issues, the input may not be split as expected. To avoid this situation it may be necessary to snap the input to the blade first, using [ST\\_Snap](#) with a small tolerance.

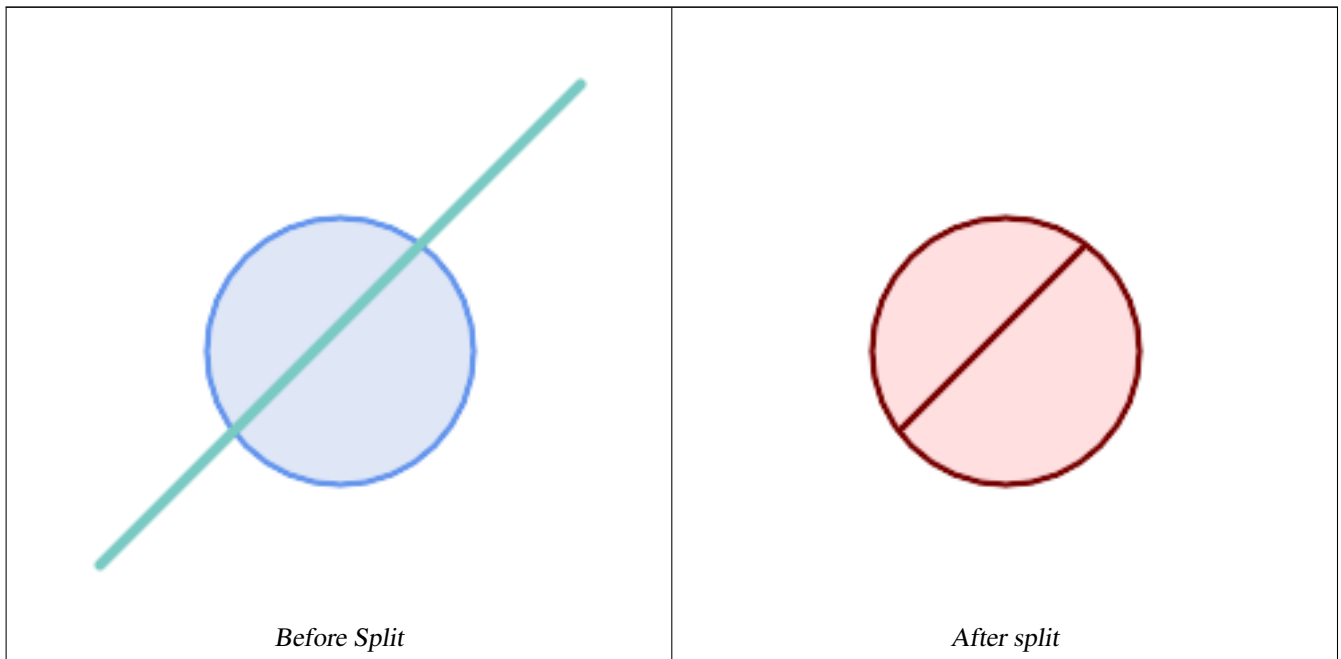
Availability: 2.0.0 requires GEOS

Enhanced: 2.2.0 support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced.

Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced.

## Examples

Split a Polygon by a Line.

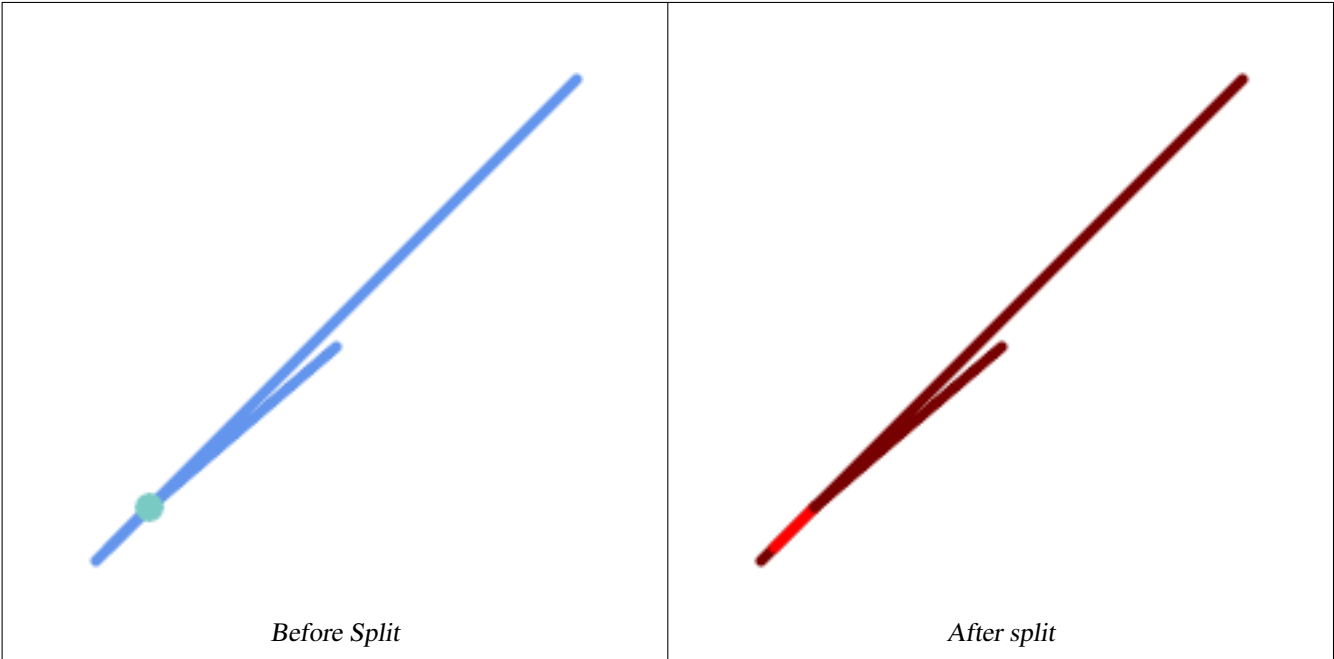


```
SELECT ST_AsText(ST_Split(
 ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50), -- circle
 ST_MakeLine(ST_Point(10, 10),ST_Point(190, 190)) -- line
));

-- result --
GEOMETRYCOLLECTION(
 POLYGON((150 90,149.039264020162 80.2454838991936,146.193976625564 ↵
 70.8658283817455,..),
 POLYGON(..)
)
```

Split a MultiLineString a Point, where the point lies exactly on both LineStrings elements.





```
SELECT ST_AsText(ST_Split(
 'MULTILINESTRING((10 10, 190 191), (15 15, 30 30, 100 90))',
 ST_Point(30,30))) As split;

split

GEOMETRYCOLLECTION(
 LINESTRING(10 10,30 30),
 LINESTRING(30 30,190 190),
 LINESTRING(15 15,30 30),
 LINESTRING(30 30,100 90)
)
```

Split a LineString by a Point, where the point does not lie exactly on the line. Shows using **ST\_Snap** to snap the line to the point to allow it to be split.

```
WITH data AS (SELECT
 'LINESTRING(0 0, 100 100)>:::geometry AS line,
 'POINT(51 50)>::: geometry AS point
)
SELECT ST_AsText(ST_Split(ST_Snap(line, point, 1), point)) AS snapped_split,
 ST_AsText(ST_Split(line, point)) AS not_snapped_not_split
FROM data;

 snapped_split | ↔
 not_snapped_not_split
-----+-----
GEOMETRYCOLLECTION(LINESTRING(0 0,51 50),LINESTRING(51 50,100 100)) | GEOMETRYCOLLECTION(↔
 LINESTRING(0 0,100 100))
```

See Also

**ST\_Snap**, **ST\_Union**

### 7.13.7 ST\_Subdivide

**ST\_Subdivide** — Computes a rectilinear subdivision of a geometry.

#### Synopsis

setof geometry **ST\_Subdivide**(geometry geom, integer max\_vertices=256, float8 gridSize = -1);

#### Description

Returns a set of geometries that are the result of dividing `geom` into parts using rectilinear lines, with each part containing no more than `max_vertices`.

`max_vertices` must be 5 or more, as 5 points are needed to represent a closed box. `gridSize` can be specified to have clipping work in fixed-precision space (requires GEOS-3.9.0+).

Point-in-polygon and other spatial operations are normally faster for indexed subdivided datasets. Since the bounding boxes for the parts usually cover a smaller area than the original geometry bbox, index queries produce fewer "hit" cases. The "hit" cases are faster because the spatial operations executed by the index recheck process fewer points.



#### Note

This is a **set-returning function** (SRF) that return a set of rows containing single geometry values. It can be used in a SELECT list or a FROM clause to produce a result set with one record for each result geometry.

---

Performed by the GEOS module.

Availability: 2.2.0

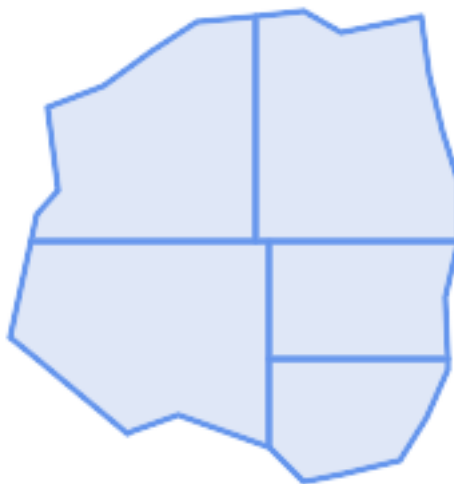
Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5.

Enhanced: 3.1.0 accept a `gridSize` parameter.

Requires GEOS  $\geq$  3.9.0 to use the `gridSize` parameter

#### Examples

**Example:** Subdivide a polygon into parts with no more than 10 vertices, and assign each part a unique id.



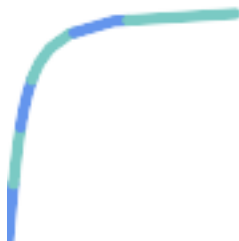
*Subdivided to maximum 10 vertices*

---

```
SELECT row_number() OVER() As rn, ST_AsText(geom) As wkt
FROM (SELECT ST_SubDivide(
 'POLYGON((132 10,119 23,85 35,68 29,66 28,49 42,32 56,22 64,32 110,40 119,36 150,
 57 158,75 171,92 182,114 184,132 186,146 178,176 184,179 162,184 141,190 122,
 190 100,185 79,186 56,186 52,178 34,168 18,147 13,132 10))'::geometry,10)) AS f(←
 geom);
```

[illegible]

**Example:** Densify a long geography line using `ST_Segmentize(geography, distance)`, and use `ST_Subdivide` to split the resulting line into sublines of 8 vertices.



*The densified and split lines.*

```
SELECT ST_AsText(ST_Subdivide(
 ST_Segmentize('LINESTRING(0 0, 85 85)::geography,
 1200000)::geometry, 8));
```

```

LINESTRING(0 0,0.487578359029357 5.57659056746196,0.984542144675897 ↵
 11.1527721155093,1.50101059639722 16.7281035483571,1.94532113630331 21.25)
LINESTRING(1.94532113630331 21.25,2.04869538062779 22.3020741387339,2.64204641967673 ↵
 27.8740533545155,3.29994062412787 33.443216802941,4.04836719489742 ↵
 39.0084282520239,4.59890468420694 42.5)
LINESTRING(4.59890468420694 42.5,4.92498503922732 44.5680389206321,5.98737409390639 ↵
 50.1195229244701,7.3290919767674 55.6587646879025,8.79638749938413 60.1969505994924)
LINESTRING(8.79638749938413 60.1969505994924,9.11375579533779 ↵
 61.1785363177625,11.6558166691368 66.6648504160202,15.642041247655 ↵
 72.0867690601745,22.8716627200212 77.3609628116894,24.6991785131552 77.8939011989848)
LINESTRING(24.6991785131552 77.8939011989848,39.4046096622744 ↵
 82.1822848017636,44.7994523421035 82.5156766227011)
LINESTRING(44.7994523421035 82.5156766227011,85 85)

```

**Example:** Subdivide the complex geometries of a table in-place. The original geometry records are deleted from the source table, and new records for each subdivided result geometry are inserted.

```
WITH complex_areas_to_subdivide AS (
 DELETE FROM polygons_table
 WHERE ST_NPoints(geom)
 > 255
 RETURNING id, column1, column2, column3, geom
)
INSERT INTO polygons_table (fid, column1, column2, column3, geom)
SELECT fid, column1, column2, column3,
 ST_Subdivide(geom, 255) AS geom
FROM complex_areas_to_subdivide;
```

**Example:** Create a new table containing subdivided geometries, retaining the key of the original geometry so that the new table can be joined to the source table. Since `ST_Subdivide` is a set-returning (table) function that returns a set of single-value rows, this syntax automatically produces a table with one row for each result part.

```
CREATE TABLE subdivided_geoms AS
SELECT pkey, ST_Subdivide(geom) AS geom
FROM original_geoms;
```

### See Also

[ST\\_ClipByBox2D](#), [ST\\_Segmentize](#), [ST\\_Split](#), [ST\\_NPoints](#)

## 7.13.8 ST\_SymDifference

`ST_SymDifference` — Computes a geometry representing the portions of geometries A and B that do not intersect.

### Synopsis

geometry **ST\_SymDifference**(geometry geomA, geometry geomB, float8 gridSize = -1);

### Description

Returns a geometry representing the portions of geometries A and B that do not intersect. This is equivalent to `ST_Union(A, B) - ST_Intersection(A, B)`. It is called a symmetric difference because `ST_SymDifference(A, B) = ST_SymDifference(B, A)`.

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

Performed by the GEOS module

Enhanced: 3.1.0 accept a `gridSize` parameter.

Requires GEOS  $\geq$  3.9.0 to use the `gridSize` parameter



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3

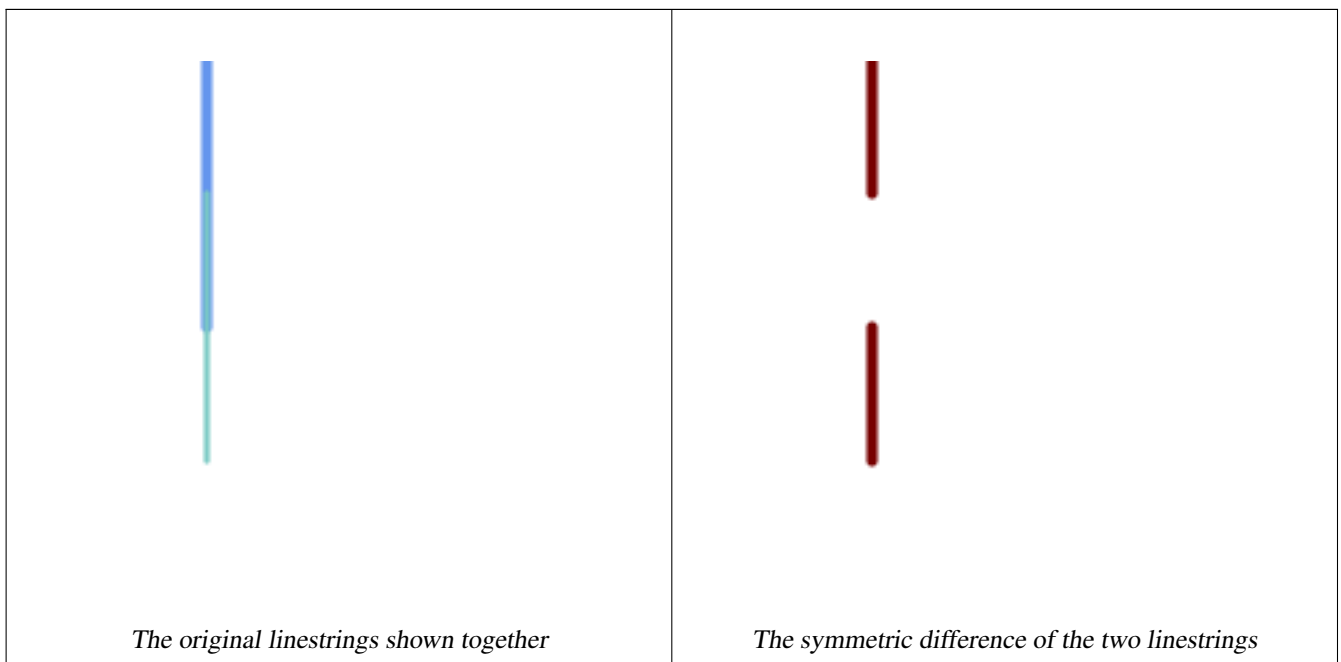


This method implements the SQL/MM specification. SQL-MM 3: 5.1.21



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

### Examples



```
--Safe for 2d - symmetric difference of 2 linestrings
SELECT ST_AsText(
 ST_SymDifference(
 ST_GeomFromText('LINESTRING(50 100, 50 200)'),
 ST_GeomFromText('LINESTRING(50 50, 50 150)')
)
);

st_astext

MULTILINESTRING((50 150,50 200),(50 50,50 100))
```

```
--When used in 3d doesn't quite do the right thing
SELECT ST_AsEWKT(ST_SymDifference(ST_GeomFromEWKT('LINESTRING(1 2 1, 1 4 2)'),
 ST_GeomFromEWKT('LINESTRING(1 1 3, 1 3 4)')));

st_astext

MULTILINESTRING((1 3 2.75,1 4 2),(1 1 3,1 2 2.25))
```

### See Also

[ST\\_Difference](#), [ST\\_Intersection](#), [ST\\_Union](#)

## 7.13.9 ST\_UnaryUnion

**ST\_UnaryUnion** — Computes the union of the components of a single geometry.

### Synopsis

geometry **ST\_UnaryUnion**(geometry geom, float8 gridSize = -1);

## Description

A single-input variant of **ST\_Union**. The input may be a single geometry, a MultiGeometry, or a GeometryCollection. The union is applied to the individual elements of the input.

This function can be used to fix MultiPolygons which are invalid due to overlapping components. However, the input components must each be valid. An invalid input component such as a bow-tie polygon may cause an error. For this reason it may be better to use **ST\_MakeValid**.

Another use of this function is to node and dissolve a collection of linestrings which cross or overlap to make them **simple**. (**ST\_Node** also does this, but it does not provide the `gridSize` option.)

It is possible to combine **ST\_Union** with **ST\_GeomCollFromText** to fine-tune how many geometries are be unioned at once. This allows trading off between memory usage and compute time, striking a balance between **ST\_Union** and **ST\_MemUnion**.

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Enhanced: 3.1.0 accept a `gridSize` parameter.

Requires GEOS >= 3.9.0 to use the `gridSize` parameter

Availability: 2.0.0

## See Also

**ST\_Union**, **ST\_MemUnion**, **ST\_MakeValid**, **ST\_GeomCollFromText**, **ST\_Node**

## 7.13.10 ST\_Union

**ST\_Union** — Computes a geometry representing the point-set union of the input geometries.

### Synopsis

```
geometry ST_Union(geometry g1, geometry g2);
geometry ST_Union(geometry g1, geometry g2, float8 gridSize);
geometry ST_Union(geometry[] g1_array);
geometry ST_Union(geometry set g1field);
geometry ST_Union(geometry set g1field, float8 gridSize);
```

### Description

Unions the input geometries, merging geometry to produce a result geometry with no overlaps. The output may be an atomic geometry, a MultiGeometry, or a Geometry Collection. Comes in several variants:

**Two-input variant:** returns a geometry that is the union of two input geometries. If either input is NULL, then NULL is returned.

**Array variant:** returns a geometry that is the union of an array of geometries.

**Aggregate variant:** returns a geometry that is the union of a rowset of geometries. The **ST\_Union()** function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the **SUM()** and **AVG()** functions do and like most aggregates, it also ignores NULL geometries.

See **ST\_UnaryUnion** for a non-aggregate, single-input variant.

The **ST\_Union** array and set variants use the fast Cascaded Union algorithm described in <http://blog.cleverelephant.ca/2009/01/-must-faster-unions-in-postgis-14.html>

A `gridSize` can be specified to work in fixed-precision space. The inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

**Note**

**ST\_GeomCollFromText** may sometimes be used in place of **ST\_Union**, if the result is not required to be non-overlapping. **ST\_Collect** is usually faster than **ST\_Union** because it performs no processing on the collected geometries.

Performed by the GEOS module.

**ST\_Union** creates **MultiLineString** and does not sew **LineStrings** into a single **LineString**. Use **ST\_LineMerge** to sew **LineStrings**.

NOTE: this function was formerly called **GeomUnion()**, which was renamed from "Union" because **UNION** is an SQL reserved word.

Enhanced: 3.1.0 accept a **gridSize** parameter.

Requires GEOS >= 3.9.0 to use the **gridSize** parameter

Changed: 3.0.0 does not depend on **SFCGAL**.

Availability: 1.4.0 - **ST\_Union** was enhanced. **ST\_Union(geomarray)** was introduced and also faster aggregate collection in PostgreSQL.



This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**. s2.1.1.3

**Note**

Aggregate version is not explicitly defined in OGC SPEC.



This method implements the SQL/MM specification. SQL-MM 3: 5.1.19 the z-index (elevation) when polygons are involved.



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

**Examples****Aggregate example**

```
SELECT id,
 ST_Union(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

**Non-Aggregate example**

```
select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(-2 3)' :: geometry))

st_astext

MULTIPOINT(-2 3,1 2)

select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(1 2)' :: geometry))

st_astext

POINT(1 2)
```

3D example - sort of supports 3D (and with mixed dimensions!)

```

select ST_AsEWKT(ST_Union(geom))
from (
 select 'POLYGON((-7 4.2,-7.1 4.2,-7.1 4.3, -7 4.2))'::geometry geom
 union all
 select 'POINT(5 5 5)'::geometry geom
 union all
 select 'POINT(-2 3 1)'::geometry geom
 union all
 select 'LINESTRING(5 5 5, 10 10 10)'::geometry geom
) as foo;

st_asewkt

GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 5,-7.1 4.2 5,-7.1 4.3 5,-7 4.2 5)));

```

### 3d example not mixing dimensions

```

select ST_AsEWKT(ST_Union(geom))
from (
 select 'POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2, -7 4.2 2))'::geometry geom
 union all
 select 'POINT(5 5 5)'::geometry geom
 union all
 select 'POINT(-2 3 1)'::geometry geom
 union all
 select 'LINESTRING(5 5 5, 10 10 10)'::geometry geom
) as foo;

st_asewkt

GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2,-7 4.2 2)));

--Examples using new Array construct
SELECT ST_Union(ARRAY(SELECT geom FROM sometable));

SELECT ST_AsText(ST_Union(ARRAY[ST_GeomFromText('LINESTRING(1 2, 3 4)'),
 ST_GeomFromText('LINESTRING(3 4, 4 5)')])) As wktunion;

--wktunion---
MULTILINESTRING((3 4,4 5),(1 2,3 4))

```

### See Also

[ST\\_GeomCollFromText](#), [ST\\_UnaryUnion](#), [ST\\_MemUnion](#), [ST\\_Intersection](#), [ST\\_Difference](#), [ST\\_SymDifference](#)

## 7.14 Procesamiento de geometría

### 7.14.1 ST\_Buffer

**ST\_Buffer** — Computes a geometry covering all points within a given distance from a geometry.



## Synopsis

```
geometry ST_Buffer(geometry g1, float radius_of_buffer, text buffer_style_parameters = "");
geometry ST_Buffer(geometry g1, float radius_of_buffer, integer num_seg_quarter_circle);
geography ST_Buffer(geography g1, float radius_of_buffer, text buffer_style_parameters);
geography ST_Buffer(geography g1, float radius_of_buffer, integer num_seg_quarter_circle);
```

## Descripción

Computes a POLYGON or MULTIPOLYGON that represents all points whose distance from a geometry/geography is less than or equal to a given distance. A negative distance shrinks the geometry rather than expanding it. A negative distance may shrink a polygon completely, in which case POLYGON EMPTY is returned. For points and lines negative distances always return empty results.

For geometry, the distance is specified in the units of the Spatial Reference System of the geometry. For geography, the distance is specified in meters.

The optional third parameter controls the buffer accuracy and style. The accuracy of circular arcs in the buffer is specified as the number of line segments used to approximate a quarter circle (default is 8). The buffer style can be specified by providing a list of blank-separated key=value pairs as follows:

- 'quad\_segs=#' : number of line segments used to approximate a quarter circle (default is 8).
- 'endcap=round|flat|square' : endcap style (defaults to "round"). 'butt' is accepted as a synonym for 'flat'.
- 'join=round|mitre|bevel' : join style (defaults to "round"). 'miter' is accepted as a synonym for 'mitre'.
- 'mitre\_limit=#.#' : mitre ratio limit (only affects mitered join style). 'miter\_limit' is accepted as a synonym for 'mitre\_limit'.
- 'side=both|left|right' : 'left' or 'right' performs a single-sided buffer on the geometry, with the buffered side relative to the direction of the line. This is only applicable to LINESTRING geometry and does not affect POINT or POLYGON geometries. By default end caps are square.

### Note



For geography this is a thin wrapper around the geometry implementation. It determines a planar spatial reference system that best fits the bounding box of the geography object (trying UTM, Lambert Azimuthal Equal Area (LAEA) North/South pole, and finally Mercator ). The buffer is computed in the planar space, and then transformed back to WGS84. This may not produce the desired behavior if the input object is much larger than a UTM zone or crosses the dateline



### Note

Buffer output is always a valid polygonal geometry. Buffer can handle invalid inputs, so buffering by distance 0 is sometimes used as a way of repairing invalid polygons. **ST\_MakeValid** can also be used for this purpose.



### Note

Buffering is sometimes used to perform a within-distance search. For this use case it is more efficient to use **ST\_DWithin**.



### Note

This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry.

Enhanced: 2.5.0 - ST\_Buffer geometry support was enhanced to allow for side buffering specification `side=both|left|right`.

Availability: 1.5 - ST\_Buffer was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added.

Realizado por el módulo GEOS.

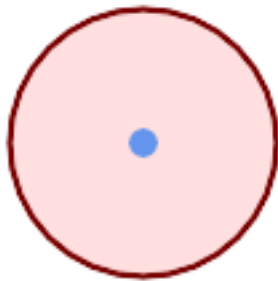


This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3



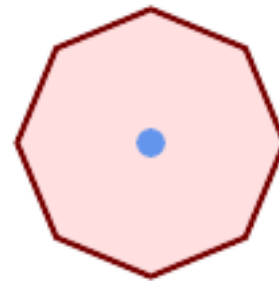
This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.30

## Ejemplos



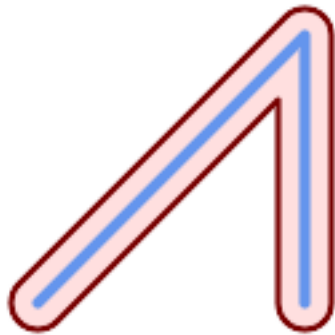
*quad\_segs=8 (por defecto)*

```
SELECT ST_Buffer(
 ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=8');
```



*quad\_segs=2 (lame)*

```
SELECT ST_Buffer(
 ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2');
```



*endcap=round join=round (default)*

```
SELECT ST_Buffer(
 ST_GeomFromText(
 'LINESTRING(50 50,150 150,150 50)'
), 10, 'endcap=round join=round');
```



*endcap=square*

```
SELECT ST_Buffer(
 ST_GeomFromText(
 'LINESTRING(50 50,150 150,150 50)'
), 10, 'endcap=square join=round');
```



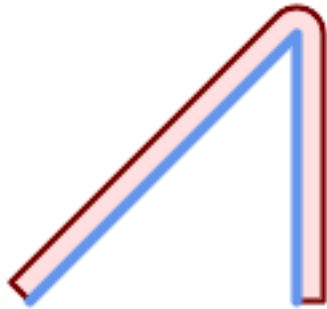
*join=bevel*

```
SELECT ST_Buffer(
 ST_GeomFromText(
 'LINESTRING(50 50,150 150,150 50)'
), 10, 'join=bevel');
```



*join=mitre mitre\_limit=5.0 (default mitre limit)*

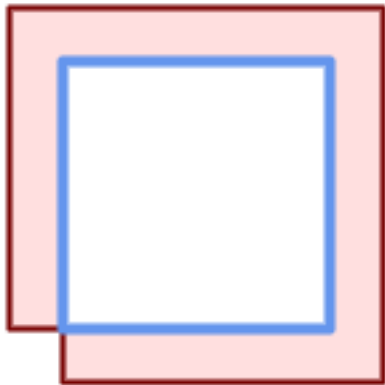
```
SELECT ST_Buffer(
 ST_GeomFromText(
 'LINESTRING(50 50,150 150,150 50)'
), 10, 'join=mitre mitre_limit=5.0');
```

*side=left*

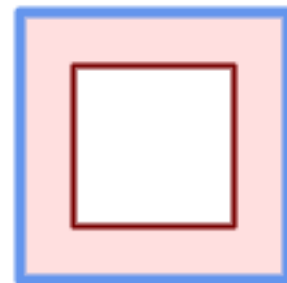
```
SELECT ST_Buffer(
 ST_GeomFromText(
 'LINESTRING(50 50,150 150,150 50)'
), 10, 'side=left');
```

*side=right*

```
SELECT ST_Buffer(
 ST_GeomFromText(
 'LINESTRING(50 50,150 150,150 50)'
), 10, 'side=right');
```

*right-hand-winding, polygon boundary side=left*

```
SELECT ST_Buffer(
 ST_ForceRHR(
 ST_Boundary(
 ST_GeomFromText(
 'POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))'
)
), 20, 'side=left');
```

*right-hand-winding, polygon boundary side=right*

```
SELECT ST_Buffer(
 ST_ForceRHR(
 ST_Boundary(
 ST_GeomFromText(
 'POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))'
)
), 20, 'side=right');
```

```
--A buffered point approximates a circle
-- A buffered point forcing approximation of (see diagram)
```

```
-- 2 points per quarter circle is poly with 8 sides (see diagram)
SELECT ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50)) As ←
 promisingcircle_pcount,
ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 2)) As lamecircle_pcount;

promisingcircle_pcount | lamecircle_pcount
-----+-----
 33 | 9

--A lighter but lamer circle
-- only 2 points per quarter circle is an octagon
--Below is a 100 meter octagon
-- Note coordinates are in NAD 83 long lat which we transform
to Mass state plane meter and then buffer to get measurements in meters;
SELECT ST_AsText(ST_Buffer(
ST_Transform(
ST_SetSRID(ST_Point(-71.063526, 42.35785), 4269), 26986)
,100,2)) As octagon;

POLYGON((236057.59057465 900908.759918696,236028.301252769 900838.049240578,235
957.59057465 900808.759918696,235886.879896532 900838.049240578,235857.59057465
900908.759918696,235886.879896532 900979.470596815,235957.59057465 901008.759918
696,236028.301252769 900979.470596815,236057.59057465 900908.759918696))
```

### Ver también

[ST\\_GeomCollFromText](#), [ST\\_DWithin](#), [ST\\_SetSRID](#), [ST\\_Transform](#), [ST\\_Union](#), [ST\\_MakeValid](#)

## 7.14.2 ST\_BuildArea

**ST\_BuildArea** — Creates a polygonal geometry formed by the linework of a geometry.

### Synopsis

geometry **ST\_BuildArea**(geometry geom);

### Descripción

Creates an areal geometry formed by the constituent linework of the input geometry. The input can be LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS, and GeometryCollections. The result is a Polygon or MultiPolygon, depending on input. If the input linework does not form polygons, NULL is returned.

This function assumes all inner geometries represent holes



#### Note

Input linework must be correctly noded for this function to work properly

Disponibilidad: 1.1.0

### Ejemplos



*These will create a donut*

```
using polygons
SELECT ST_BuildArea(ST_Collect(smallc,bigc))
FROM (SELECT
 ST_Buffer(
 ST_GeomFromText('POINT(100 90)'), 25) As smallc,
 ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50) As bigc) As foo;

using linestrings
SELECT ST_BuildArea(ST_Collect(smallc,bigc))
FROM (SELECT
 ST_ExteriorRing(ST_Buffer(
 ST_GeomFromText('POINT(100 90)'), 25)) As smallc,
 ST_ExteriorRing(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50)) As bigc) As foo;
```

#### Ver también

[ST\\_Node](#), [ST\\_MakePolygon](#), [ST\\_MakeValid](#), [ST\\_BdPolyFromText](#), [ST\\_BdMPolyFromText](#) (wrappers to this function with standard OGC interface)

### 7.14.3 ST\_Centroid

**ST\_Centroid** — Returns the geometric center of a geometry.

#### Synopsis

```
geometry ST_Centroid(geometry g1);
geography ST_Centroid(geography g1, boolean use_spheroid = true);
```

#### Descripción

Computes a point which is the geometric center of mass of a geometry. For [MULTI]POINTS, the centroid is the arithmetic mean of the input coordinates. For [MULTI]LINESTRINGS, the centroid is computed using the weighted length of each line segment. For [MULTI]POLYGONS, the centroid is computed in terms of area. If an empty geometry is supplied, an empty

GEOMETRYCOLLECTION is returned. If NULL is supplied, NULL is returned. If CIRCULARSTRING or COMPOUNDCURVE are supplied, they are converted to linestring with CurveToLine first, then same than for LINESTRING

For mixed-dimension input, the result is equal to the centroid of the component Geometries of highest dimension (since the lower-dimension geometries contribute zero "weight" to the centroid).

Note that for polygonal geometries the centroid does not necessarily lie in the interior of the polygon. For example, see the diagram below of the centroid of a C-shaped polygon. To construct a point guaranteed to lie in the interior of a polygon use [ST\\_PointOnSurface](#).

New in 2.3.0 : supports CIRCULARSTRING and COMPOUNDCURVE (using CurveToLine)

Availability: 2.4.0 support for geography was introduced.



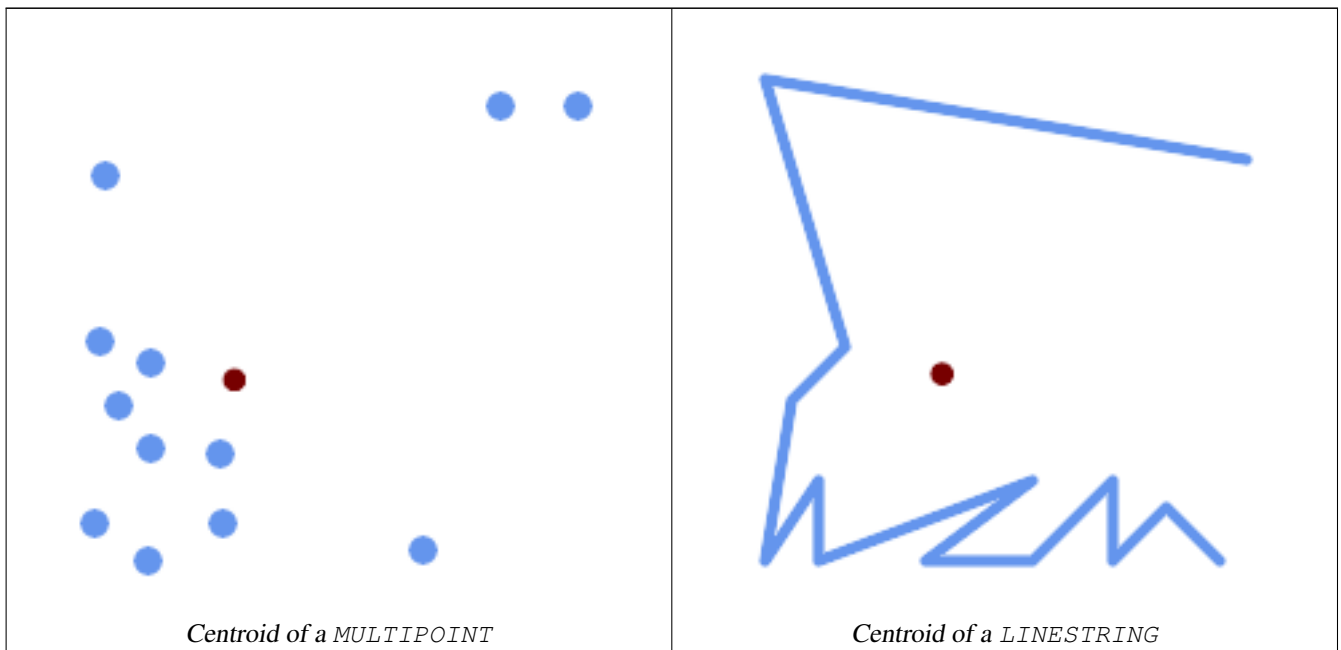
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).

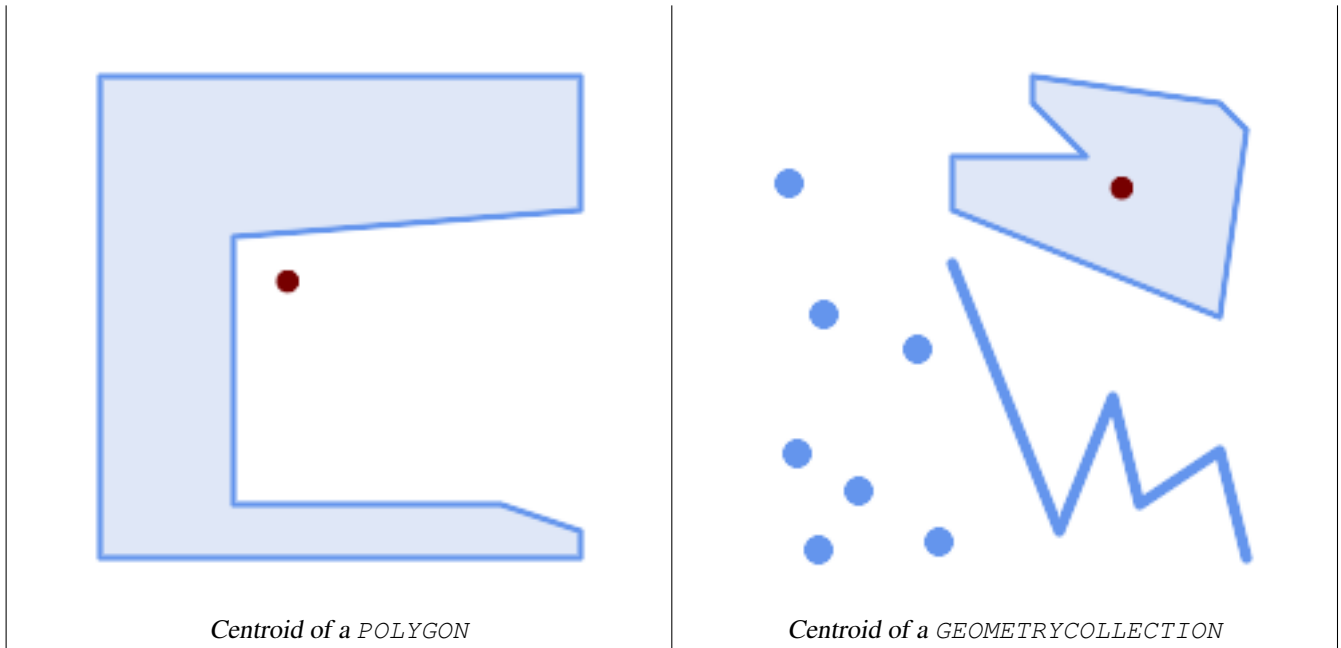


This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5

## Ejemplos

In the following illustrations the red dot is the centroid of the source geometry.





```

SELECT ST_AsText(ST_Centroid('MULTIPOINT (-1 0, -1 2, -1 3, -1 4, -1 7, 0 1, 0 3, 1 1, 2 0, 6 0, 7 8, 9 8, 10 6)'));
 st_astext

POINT(2.30769230769231 3.30769230769231)
(1 row)

SELECT ST_AsText(ST_centroid(g))
FROM ST_GeomFromText('CIRCULARSTRING(0 2, -1 1,0 0, 0.5 0, 1 0, 2 1, 1 2, 0.5 2, 0 2)') AS g ;

POINT(0.5 1)

SELECT ST_AsText(ST_centroid(g))
FROM ST_GeomFromText('COMPOUNDCURVE(CIRCULARSTRING(0 2, -1 1,0 0),(0 0, 0.5 0, 1 0), CIRCULARSTRING(1 0, 2 1, 1 2),(1 2, 0.5 2, 0 2))') AS g;

POINT(0.5 1)

```

#### Ver también

[ST\\_PointOnSurface](#), [ST\\_GeometricMedian](#)

### 7.14.4 ST\_ChaikinSmoothing

**ST\_ChaikinSmoothing** — Returns a smoothed version of a geometry, using the Chaikin algorithm

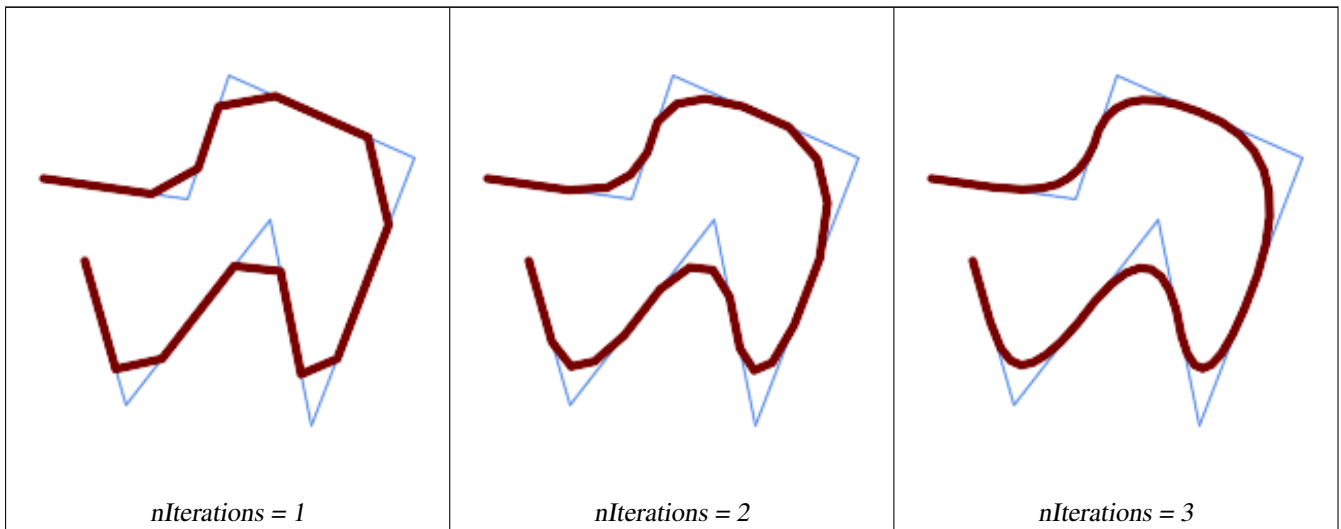
#### Synopsis

geometry **ST\_ChaikinSmoothing**(geometry geom, integer nIterations = 1, boolean preserveEndpoints = false);





Smoothing a LineString using 1, 2 and 3 iterations:



```
SELECT ST_ChaikinSmoothing(
 'LINESTRING (10 140, 80 130, 100 190, 190 150, 140 20, 120 120, 50 30, 30 100)' ←
 ,
 generate_series(1, 3));
```

**Ver también**

[ST\\_Simplify](#), [ST\\_SimplifyPreserveTopology](#), [ST\\_SimplifyVW](#)

### 7.14.5 ST\_ConcaveHull

**ST\_ConcaveHull** — Computes a possibly concave geometry that contains all input geometry vertices

#### Synopsis

geometry **ST\_ConcaveHull**(geometry param\_geom, float param\_pctconvex, boolean param\_allow\_holes = false);

#### Descripción

A concave hull is a (usually) concave geometry which contains the input, and whose vertices are a subset of the input vertices. In the general case the concave hull is a Polygon. The concave hull of two or more collinear points is a two-point LineString. The concave hull of one or more identical points is a Point. The polygon will not contain holes unless the optional `param_allow_holes` argument is specified as true.

One can think of a concave hull as "shrink-wrapping" a set of points. This is different to the [convex hull](#), which is more like wrapping a rubber band around the points. A concave hull generally has a smaller area and represents a more natural boundary for the input points.

The `param_pctconvex` controls the concaveness of the computed hull. A value of 1 produces the convex hull. Values between 1 and 0 produce hulls of increasing concaveness. A value of 0 produces a hull with maximum concaveness (but still a single polygon). Choosing a suitable value depends on the nature of the input data, but often values between 0.3 and 0.1 produce reasonable results.

**Note**

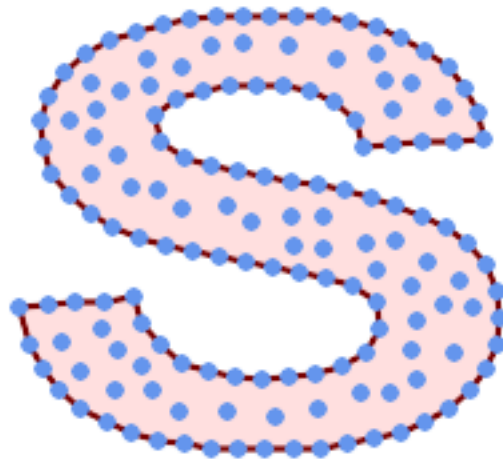
Technically, the `param_pctconvex` determines a length as a fraction of the difference between the longest and shortest edges in the Delaunay Triangulation of the input points. Edges longer than this length are "eroded" from the triangulation. The triangles remaining form the concave hull.

For point and linear inputs, the hull will enclose all the points of the inputs. For polygonal inputs, the hull will enclose all the points of the input *and also* all the areas covered by the input. If you want a point-wise hull of a polygonal input, convert it to points first using [ST\\_Points](#).

This is not an aggregate function. To compute the concave hull of a set of geometries use [ST\\_GeomCollFromText](#) (e.g. `ST_ConcaveHull( ST_Collect( geom ), 0.80)`).

Disponibilidad: 2.0.0

Enhanced: 3.3.0, GEOS native implementation enabled for GEOS 3.11+

**Ejemplos**

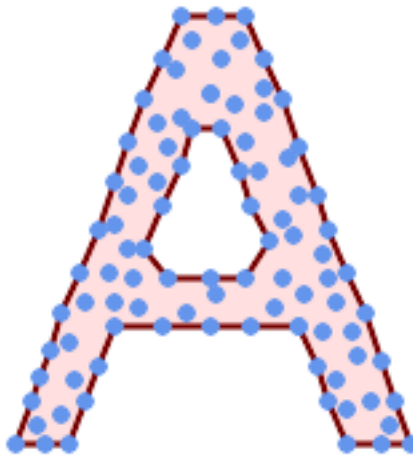
*Concave Hull of a MultiPoint*

```
SELECT ST_AsText(ST_ConcaveHull(
 'MULTIPOINT ((10 72), (53 76), (56 66), (63 58), (71 51), (81 48), (91 46), (101 45), (111 46), (121 47), (131 50), (140 55), (145 64), (144 74), (135 80), (125 83), (115 85), (105 87), (95 89), (85 91), (75 93), (65 95), (55 98), (45 102), (37 107), (29 114), (22 122), (19 132), (18 142), (21 151), (27 160), (35 167), (44 172), (54 175), (64 178), (74 180), (84 181), (94 181), (104 181), (114 181), (124 181), (134 179), (144 177), (153 173), (162 168), (171 162), (177 154), (182 145), (184 135), (139 132), (136 142), (128 149), (119 153), (109 155), (99 155), (89 155), (79 153), (69 150), (61 144), (63 134), (72 128), (82 125), (92 123), (102 121), (112 119), (122 118), (132 116), (142 113), (151 110), (161 106), (170 102), (178 96), (185 88), (189 78), (190 68), (189 58), (185 49), (179 41), (171 34), (162 29), (153 25), (143 23), (133 21), (123 19), (113 19), (102 19), (92 19), (82 19), (72 21), (62 22), (52 25), (43 29), (33 34), (25 41), (19 49), (14 58), (21 73), (31 74), (42 74), (173 134), (161 134), (150 133), (97 104), (52 117), (157 156), (94 171), (112 106), (169 73), (58 165), (149 40), (70 33), (147 157), (48 153), (140 96), (47 129), (173 55), (144 86), (159 67), (150 146), (38 136), (111 170), (124 94), (26 59), (60 41), (71 162), (41 64), (88 110), (122 34), (151 97), (157 56), (39 146), (88 33), (159 45), (47 56), (138 40), (129 165), (33 48), (106 31), (169 147), (37 122), (71 109), (163 89), (37 156), (82 170), (180 72), (29 142), (46 41), (59 155), (124 106), (157 80), (175 82), (56 50), (62 116), (113 95), (144 167))',
```

```

 0.1));
---st_astext---
POLYGON ((18 142, 21 151, 27 160, 35 167, 44 172, 54 175, 64 178, 74 180, 84 181, 94 181, ↵
104 181, 114 181, 124 181, 134 179, 144 177, 153 173, 162 168, 171 162, 177 154, 182 ↵
145, 184 135, 173 134, 161 134, 150 133, 139 132, 136 142, 128 149, 119 153, 109 155, 99 ↵
155, 89 155, 79 153, 69 150, 61 144, 63 134, 72 128, 82 125, 92 123, 102 121, 112 119, ↵
122 118, 132 116, 142 113, 151 110, 161 106, 170 102, 178 96, 185 88, 189 78, 190 68, ↵
189 58, 185 49, 179 41, 171 34, 162 29, 153 25, 143 23, 133 21, 123 19, 113 19, 102 19, ↵
92 19, 82 19, 72 21, 62 22, 52 25, 43 29, 33 34, 25 41, 19 49, 14 58, 10 72, 21 73, 31 ↵
74, 42 74, 53 76, 56 66, 63 58, 71 51, 81 48, 91 46, 101 45, 111 46, 121 47, 131 50, 140 ↵
55, 145 64, 144 74, 135 80, 125 83, 115 85, 105 87, 95 89, 85 91, 75 93, 65 95, 55 98, ↵
45 102, 37 107, 29 114, 22 122, 19 132, 18 142))

```

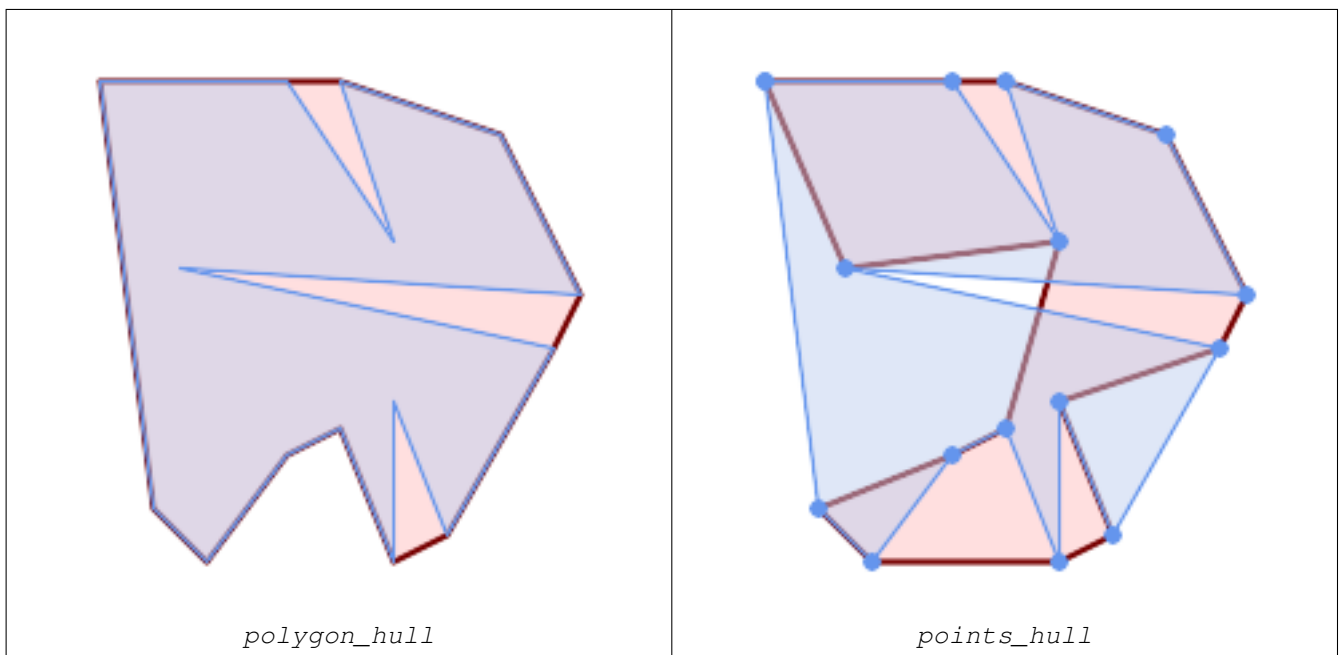


*Concave Hull of a MultiPoint, allowing holes*

```

SELECT ST_AsText(ST_ConcaveHull(
 'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 36), (46 ↵
20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 100), (63 118), ↵
(68 133), (74 149), (81 164), (88 180), (101 180), (112 180), (119 164), (126 ↵
149), (132 131), (139 113), (143 100), (150 84), (157 69), (163 51), (168 36), ↵
(174 20), (163 20), (150 20), (143 36), (139 49), (132 64), (99 151), (92 138), ↵
(88 124), (81 109), (74 93), (70 82), (83 82), (99 82), (112 82), (126 82), (121 ↵
96), (114 109), (110 122), (103 138), (99 151), (34 27), (43 31), (48 44), (46 ↵
58), (52 73), (63 73), (61 84), (72 71), (90 69), (101 76), (123 71), (141 62), ↵
(166 27), (150 33), (159 36), (146 44), (154 53), (152 62), (146 73), (134 76), ↵
(143 82), (141 91), (130 98), (126 104), (132 113), (128 127), (117 122), (112 ↵
133), (119 144), (108 147), (119 153), (110 171), (103 164), (92 171), (86 160), ↵
(88 142), (79 140), (72 124), (83 131), (79 118), (68 113), (63 102), (68 93), ↵
(35 45))',
 0.15, true));
---st_astext---
POLYGON ((43 69, 50 84, 57 100, 63 118, 68 133, 74 149, 81 164, 88 180, 101 180, 112 180, ↵
119 164, 126 149, 132 131, 139 113, 143 100, 150 84, 157 69, 163 51, 168 36, 174 20, 163 ↵
20, 150 20, 143 36, 139 49, 132 64, 114 64, 99 64, 81 64, 63 64, 57 49, 52 36, 46 20, ↵
37 20, 26 20, 32 36, 35 45, 39 55, 43 69), (88 124, 81 109, 74 93, 83 82, 99 82, 112 82, ↵
121 96, 114 109, 110 122, 103 138, 92 138, 88 124))

```



Comparing a concave hull of a Polygon to the concave hull of the constituent points. The hull respects the boundary of the polygon, whereas the points-based hull does not.

```
WITH data(geom) AS (VALUES
 ('POLYGON ((10 90, 39 85, 61 79, 50 90, 80 80, 95 55, 25 60, 90 45, 70 16, 63 38, 60 10, ↵
 50 30, 43 27, 30 10, 20 20, 10 90))'::geometry)
)
SELECT ST_ConcaveHull(geom, 0.1) AS polygon_hull,
 ST_ConcaveHull(ST_Points(geom), 0.1) AS points_hull
FROM data;
```

Using with `ST_Collect` to compute the concave hull of a geometry set.

```
-- Compute estimate of infected area based on point observations
SELECT disease_type,
 ST_ConcaveHull(ST_Collect(obs_pnt), 0.3) AS geom
FROM disease_obs
GROUP BY disease_type;
```

## Ver también

[ST\\_ConvexHull](#), [ST\\_GeomCollFromText](#), [ST\\_AlphaShape](#), [ST\\_OptimalAlphaShape](#)

## 7.14.6 ST\_ConvexHull

`ST_ConvexHull` — Computes the convex hull of a geometry.

### Synopsis

geometry **ST\_ConvexHull**(geometry geomA);

## Descripción

Computes the convex hull of a geometry. The convex hull is the smallest convex geometry that encloses all geometries in the input.

One can think of the convex hull as the geometry obtained by wrapping an rubber band around a set of geometries. This is different from a **concave hull** which is analogous to "shrink-wrapping" the geometries. A convex hull is often used to determine an affected area based on a set of point observations.

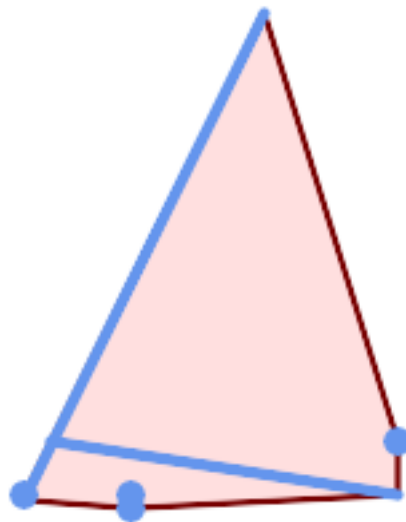
In the general case the convex hull is a Polygon. The convex hull of two or more collinear points is a two-point LineString. The convex hull of one or more identical points is a Point.

This is not an aggregate function. To compute the convex hull of a set of geometries, use **ST\_GeomCollFromText** to aggregate them into a geometry collection (e.g. `ST_ConvexHull(ST_Collect (geom) )`).

Realizado por el módulo de GEOS

- ✓ This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**. s2.1.1.3
- ✓ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.16
- ✓ This function supports 3d and will not drop the z-index.

## Ejemplos



*Convex Hull of a MultiLinestring and a MultiPoint*

```
SELECT ST_AsText (ST_ConvexHull (
 ST_Collect (
 ST_GeomFromText ('MULTILINESTRING((100 190,10 8),(150 10, 20 30))'),
 ST_GeomFromText ('MULTIPOINT(50 5, 150 30, 50 10, 10 10)')
)));
---st_astext---
POLYGON((50 5,10 8,10 10,100 190,150 30,150 10,50 5))
```

Using with **ST\_Collect** to compute the convex hulls of geometry sets.

```
--Get estimate of infected area based on point observations
SELECT d.disease_type,
 ST_ConvexHull(ST_Collect(d.geom)) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```

**Ver también**

[ST\\_GeomCollFromText](#), [ST\\_ConcaveHull](#), [ST\\_MinimumBoundingCircle](#)

### 7.14.7 ST\_DelaunayTriangles

ST\_DelaunayTriangles — Returns the Delaunay triangulation of the vertices of a geometry.

**Synopsis**

geometry **ST\_DelaunayTriangles**(geometry g1, float tolerance = 0.0, int4 flags = 0);

**Descripción**

Computes the **Delaunay triangulation** of the vertices of the input geometry. The optional `tolerance` can be used to snap nearby input vertices together, which improves robustness in some situations. The result geometry is bounded by the convex hull of the input vertices. The result geometry representation is determined by the `flags` code:

- 0 - a GEOMETRYCOLLECTION of triangular POLYGONS (default)
- 1 - a MULTILINESTRING of the edges of the triangulation
- 2 - A TIN of the triangulation

Realizado por el módulo GEOS.

Disponibilidad: 2.1.0



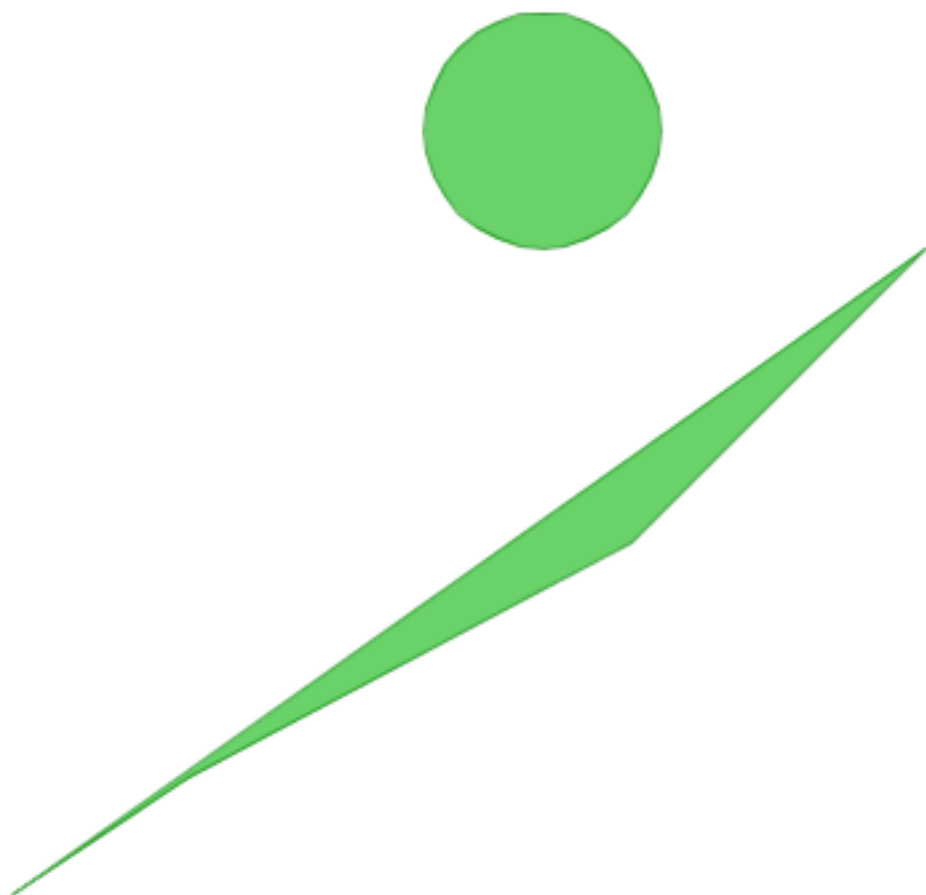
This function supports 3d and will not drop the z-index.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**Ejemplos**

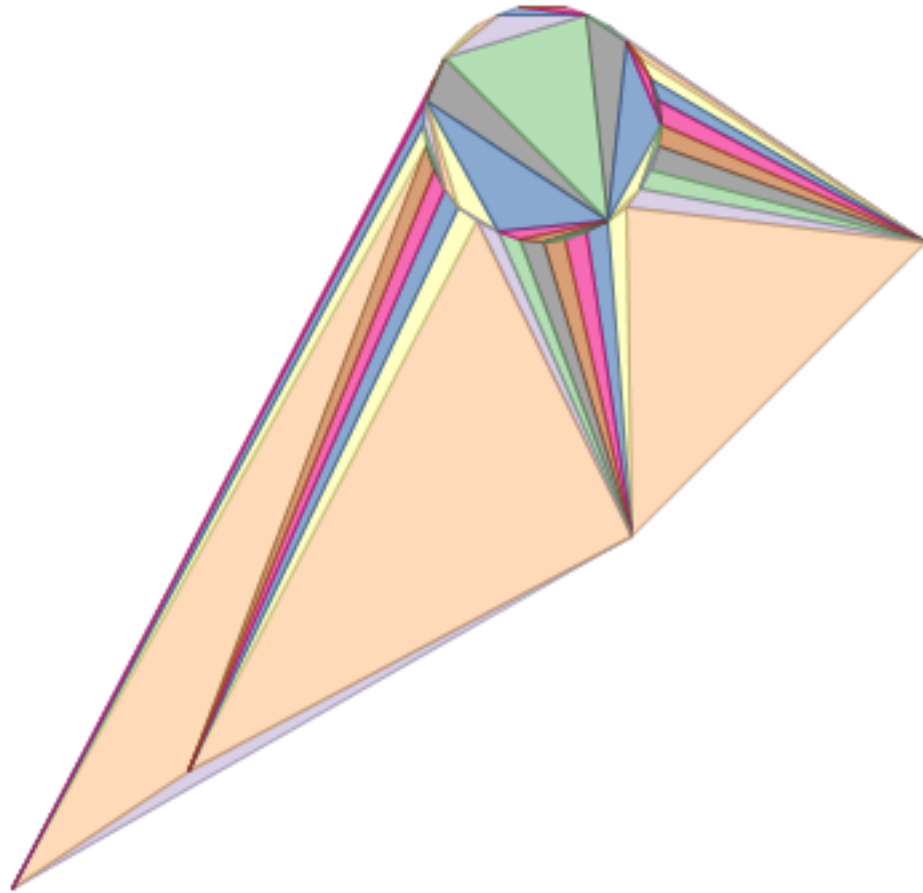
---



*Original polygons*

```
our original geometry
ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
 50 60, 125 100, 175 150))'),
 ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
)
```

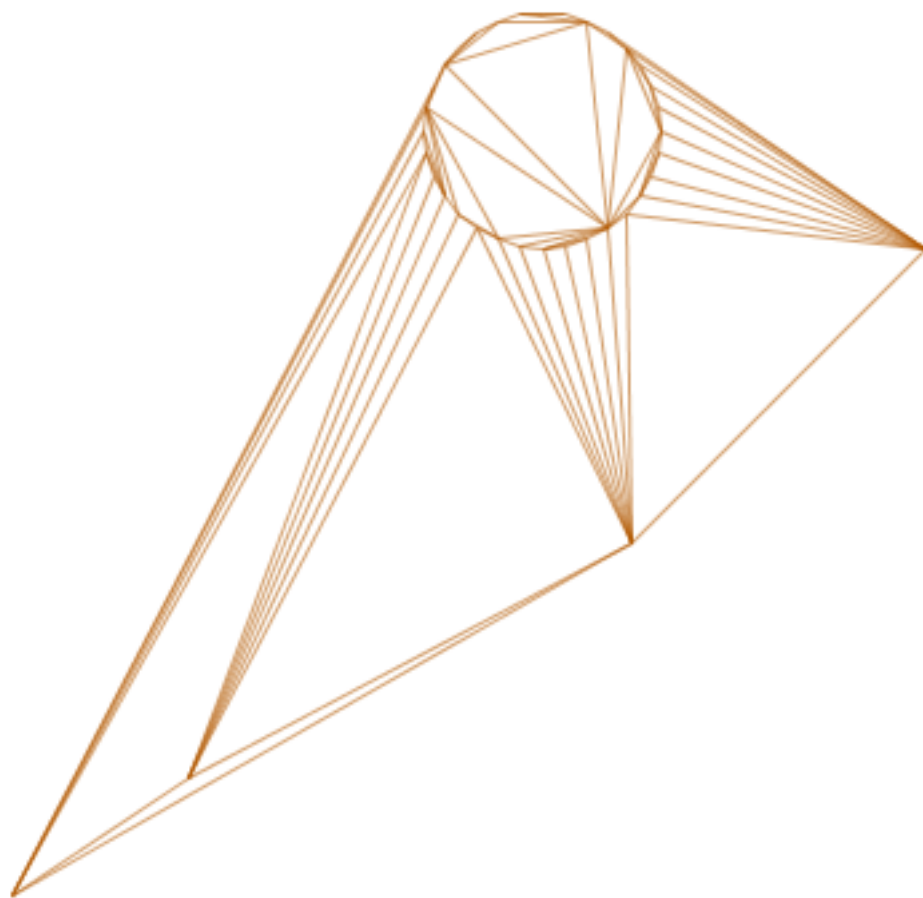




*ST\_DelaunayTriangles of 2 polygons: delaunay triangle polygons each triangle themed in different color*

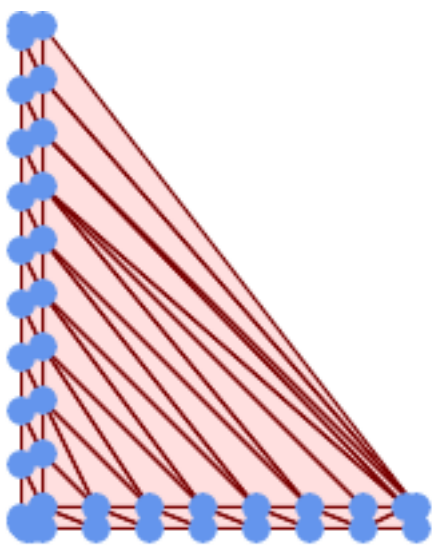
geometries overlaid multilinestring triangles

```
SELECT
 ST_DelaunayTriangles(
 ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
 50 60, 125 100, 175 150))'),
 ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
)
 As dtriag;
```



*-- delaunay triangles as multilinestring*

```
SELECT
 ST_DelaunayTriangles(
 ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
 50 60, 125 100, 175 150))'),
 ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
),0.001,1)
 As dtriag;
```



-- delaunay triangles of 45 points as 55 triangle polygons

this produces a table of 42 points that form an L shape

```
SELECT (ST_DumpPoints(ST_GeomFromText (
'MULTIPOINT(14 14,34 14,54 14,74 14,94 14,114 14,134 14,
150 14,154 14,154 6,134 6,114 6,94 6,74 6,54 6,34 6,
14 6,10 6,8 6,7 7,6 8,6 10,6 30,6 50,6 70,6 90,6 110,6 130,
6 150,6 170,6 190,6 194,14 194,14 174,14 154,14 134,14 114,
14 94,14 74,14 54,14 34,14 14)'))).geom
 INTO TABLE l_shape;
```

output as individual polygon triangles

```
SELECT ST_AsText((ST_Dump(geom)).geom) As wkt
FROM (SELECT ST_DelaunayTriangles(ST_Collect(geom)) As geom
FROM l_shape) As foo;
```

wkt

```
POLYGON((6 194,6 190,14 194,6 194))
POLYGON((14 194,6 190,14 174,14 194))
POLYGON((14 194,14 174,154 14,14 194))
POLYGON((154 14,14 174,14 154,154 14))
POLYGON((154 14,14 154,150 14,154 14))
POLYGON((154 14,150 14,154 6,154 14))
```

### Example using vertices with Z values.

3D multipoint

```
SELECT ST_AsText(ST_DelaunayTriangles(ST_GeomFromText (
'MULTIPOINT Z(14 14 10, 150 14 100,34 6 25, 20 10 150)')))) As wkt;
```

wkt

```
GEOMETRYCOLLECTION Z (POLYGON Z ((14 14 10,20 10 150,34 6 25,14 14 10))
,POLYGON Z ((14 14 10,34 6 25,150 14 100,14 14 10)))
```

**Ver también**

[ST\\_VoronoiPolygons](#), [ST\\_TriangulatePolygon](#), [ST\\_ConstrainedDelaunayTriangles](#), [ST\\_VoronoiLines](#), [ST\\_ConvexHull](#)

**7.14.8 ST\_FilterByM**

**ST\_FilterByM** — Removes vertices based on their M value

**Synopsis**

geometry **ST\_FilterByM**(geometry geom, double precision min, double precision max = null, boolean returnM = false);

**Descripción**

Filters out vertex points based on their M-value. Returns a geometry with only vertex points that have a M-value larger or equal to the min value and smaller or equal to the max value. If max-value argument is left out only min value is considered. If fourth argument is left out the m-value will not be in the resulting geometry. If resulting geometry have too few vertex points left for its geometry type an empty geometry will be returned. In a geometry collection geometries without enough points will just be left out silently.

This function is mainly intended to be used in conjunction with `ST_SetEffectiveArea`. `ST_EffectiveArea` sets the effective area of a vertex in its m-value. With `ST_FilterByM` it then is possible to get a simplified version of the geometry without any calculations, just by filtering

**Note**

There is a difference in what `ST_SimplifyVW` returns when not enough points meet the criteria compared to `ST_FilterByM`. `ST_SimplifyVW` returns the geometry with enough points while `ST_FilterByM` returns an empty geometry

**Note**

Note that the returned geometry might be invalid

**Note**

This function returns all dimensions, including the Z and M values

Availability: 2.5.0

**Ejemplos**

A linestring is filtered

```
SELECT ST_AsText(ST_FilterByM(geom,30)) simplified
FROM (SELECT ST_SetEffectiveArea('LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry') geom ←
) As foo;

result

 simplified

LINESTRING(5 2,7 25,10 10)
```

Ver también

[ST\\_SetEffectiveArea](#), [ST\\_SimplifyVW](#)

### 7.14.9 ST\_GeneratePoints

ST\_GeneratePoints — Generates random points contained in a Polygon or MultiPolygon.

#### Synopsis

```
geometry ST_GeneratePoints(g geometry , npoints integer);
geometry ST_GeneratePoints(geometry g , integer npoints , integer seed = 0);
```

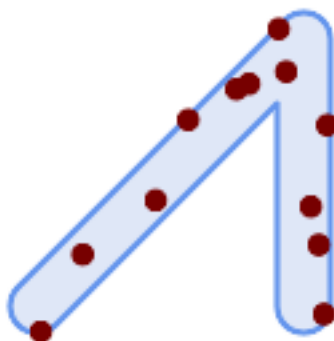
#### Descripción

ST\_GeneratePoints generates a given number of pseudo-random points which lie within the input area. The optional `seed` is used to regenerate a deterministic sequence of points, and must be greater than zero.

Disponibilidad: 2.3.0

Enhanced: 3.0.0, added seed parameter

#### Ejemplos



*Generated 12 Points overlaid on top of original polygon using a random seed value 1996*

```
SELECT ST_GeneratePoints(geom, 12, 1996)
FROM (
 SELECT ST_Buffer(
 ST_GeomFromText(
 'LINESTRING(50 50,150 150,150 50)'),
 10, 'endcap=round join=round') AS geom
) AS s;
```

### 7.14.10 ST\_GeometricMedian

ST\_GeometricMedian — Returns the geometric median of a MultiPoint.

Synopsis

geometry **ST\_GeometricMedian** ( geometry geom, float8 tolerance = NULL, int max\_iter = 10000, boolean fail\_if\_not\_converged = false);

Descripción

Computes the approximate geometric median of a MultiPoint geometry using the Weiszfeld algorithm. The geometric median is the point minimizing the sum of distances to the input points. It provides a centrality measure that is less sensitive to outlier points than the centroid (center of mass).

The algorithm iterates until the distance change between successive iterations is less than the supplied `tolerance` parameter. If this condition has not been met after `max_iterations` iterations, the function produces an error and exits, unless `fail_if_not_converged` is set to `false` (the default).

If a `tolerance` argument is not provided, the tolerance value is calculated based on the extent of the input geometry.

If present, the input point M values are interpreted as their relative weights.

Disponibilidad: 2.3.0

Enhanced: 2.5.0 Added support for M as weight of points.

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

Ejemplos



Comparison of the geometric median (red) and centroid (turquoise) of a MultiPoint.

```
WITH test AS (
SELECT 'MULTIPOINT((10 10), (10 40), (40 10), (190 190))'::geometry geom
SELECT
 ST_AsText(ST_Centroid(geom)) centroid,
 ST_AsText(ST_GeometricMedian(geom)) median
FROM test;

 centroid | median
-----+-----
POINT(62.5 62.5) | POINT(25.01778421249728 25.01778421249728)
(1 row)
```

Ver también

[ST\\_Centroid](#)

### 7.14.11 ST\_LineMerge

ST\_LineMerge — Return the lines formed by sewing together a MultiLineString.

#### Synopsis

```
geometry ST_LineMerge(geometry amultilinestring);
geometry ST_LineMerge(geometry amultilinestring, boolean directed);
```

#### Descripción

Returns a LineString or MultiLineString formed by joining together the line elements of a MultiLineString. Lines are joined at their endpoints at 2-way intersections. Lines are not joined across intersections of 3-way or greater degree.

If **directed** is TRUE, then ST\_LineMerge will not change point order within LineStrings, so lines with opposite directions will not be merged



#### Note

Only use with MultiLineString/LineStrings. Other geometry types return an empty GeometryCollection

---

Realizado por el módulo GEOS.

Enhanced: 3.3.0 accept a directed parameter.

Requires GEOS >= 3.11.0 to use the directed parameter.

Disponibilidad: 1.1.0

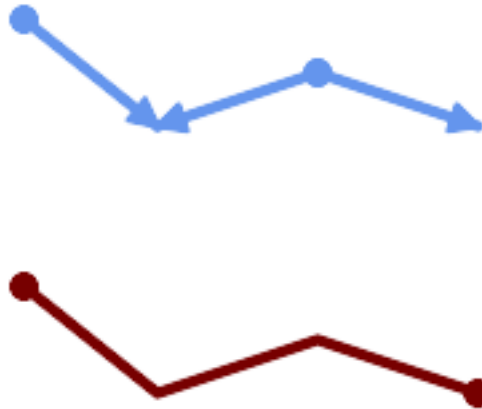


#### Warning

This function strips the M dimension.

---

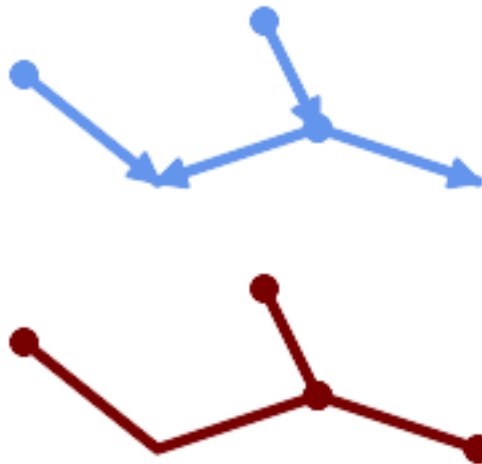
## Ejemplos



*Merging lines with different orientation.*

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120))'
));

LINESTRING(10 160,60 120,120 140,180 120)
```



*Lines are not merged across intersections with degree > 2.*

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120), (100 180, 120 140))'
));

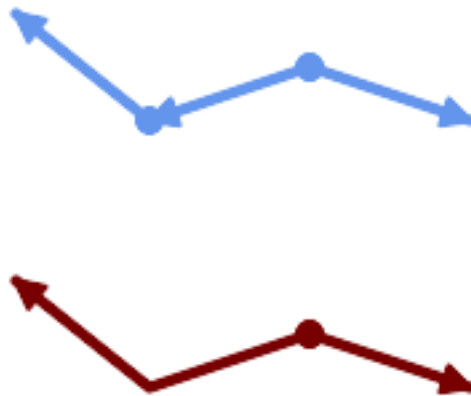
MULTILINESTRING((10 160,60 120,120 140),(100 180,120 140),(120 140,180 120))
```

If merging is not possible due to non-touching lines, the original MultiLineString is returned.



```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33), (-45.2 -33.2,-46 -32)) '
));

MULTILINESTRING((-45.2 -33.2,-46 -32), (-29 -27,-30 -29.7,-36 -31,-45 -33))
```



*Lines with opposite directions are not merged if directed = TRUE.*

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((60 30, 10 70), (120 50, 60 30), (120 50, 180 30))',
TRUE));

MULTILINESTRING((120 50,60 30,10 70), (120 50,180 30))
```

Example showing Z-dimension handling.

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 6), (-29 -27 12,-30 -29.7 5), (-45 -33 1,-46 -32 11)) '
));

LINESTRING Z (-30 -29.7 5,-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 1,-46 -32 11)
```

**Ver también**

[ST\\_Segmentize](#), [ST\\_LineSubstring](#)

### 7.14.12 ST\_MaximumInscribedCircle

**ST\_MaximumInscribedCircle** — Computes the largest circle contained within a geometry.

#### Synopsis

(geometry, geometry, double precision) **ST\_MaximumInscribedCircle**(geometry geom);

Descripción

Finds the largest circle that is contained within a (multi)polygon, or which does not overlap any lines and points. Returns a record with fields:

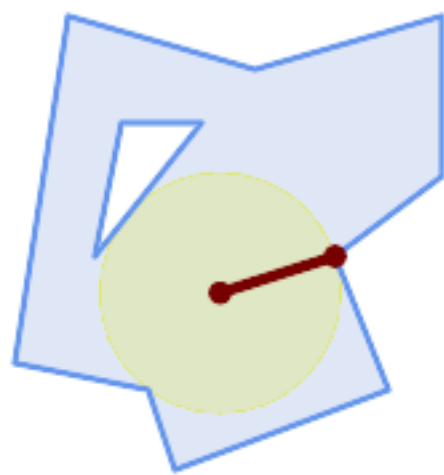
- center - center point of the circle
- nearest - a point on the geometry nearest to the center
- radius - radius of the circle

For polygonal inputs, the circle is inscribed within the boundary rings, using the internal rings as boundaries. For linear and point inputs, the circle is inscribed within the convex hull of the input, using the input lines and points as further boundaries.

Availability: 3.1.0.

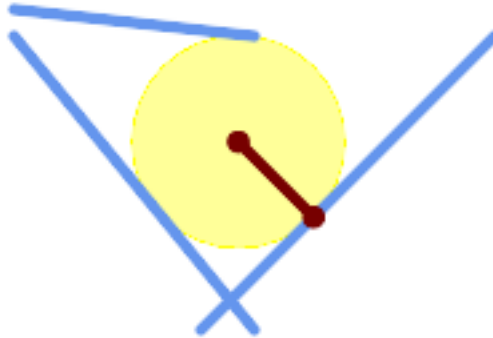
Requires GEOS >= 3.9.0.

Ejemplos



Maximum inscribed circle of a polygon. Center, nearest point, and radius are returned.

```
SELECT radius, ST_AsText(center) AS center, ST_AsText(nearest) AS nearest
FROM ST_MaximumInscribedCircle(
 'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 50, 40 180),
 (60 140, 50 90, 90 140, 60 140))');
radius | center | nearest
-----+-----+-----
45.165845650018 | POINT(96.953125 76.328125) | POINT(140 90)
```



*Maximum inscribed circle of a multi-linestring. Center, nearest point, and radius are returned.*

**Ver también**

[ST\\_MinimumBoundingRadius](#), [ST\\_LargestEmptyCircle](#)

### 7.14.13 ST\_LargestEmptyCircle

`ST_LargestEmptyCircle` — Computes the largest circle not overlapping a geometry.

#### Synopsis

(geometry, geometry, double precision) **ST\_LargestEmptyCircle**(geometry geom, double precision tolerance=0.0, geometry boundary=POINT EMPTY);

#### Descripción

Finds the largest circle which does not overlap a set of point and line obstacles. (Polygonal geometries may be included as obstacles, but only their boundary lines are used.) The center of the circle is constrained to lie inside a polygonal boundary, which by default is the convex hull of the input geometry. The circle center is the point in the interior of the boundary which has the farthest distance from the obstacles. The circle itself is provided by the center point and a nearest point lying on an obstacle determining the circle radius.

The circle center is determined to a given accuracy specified by a distance tolerance, using an iterative algorithm. If the accuracy distance is not specified a reasonable default is used.

Returns a record with fields:

- `center` - center point of the circle
- `nearest` - a point on the geometry nearest to the center
- `radius` - radius of the circle

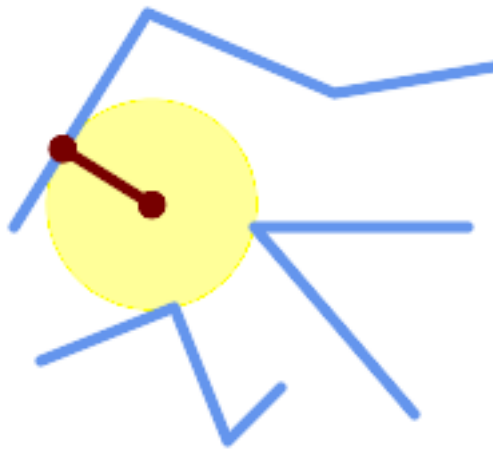
To find the largest empty circle in the interior of a polygon, see [ST\\_MaximumInscribedCircle](#).

Availability: 3.4.0.

Requires GEOS >= 3.9.0.

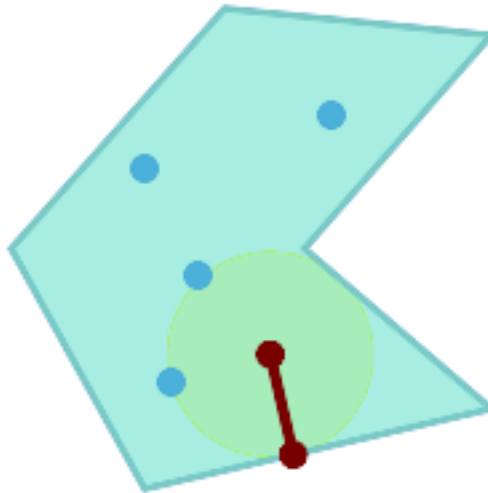
## Ejemplos

```
SELECT radius,
 ST_AsText(center) AS center,
 ST_AsText(nearest) AS nearest
FROM ST_LargestEmptyCircle(
 'MULTILINESTRING (
 (10 100, 60 180, 130 150, 190 160),
 (20 50, 70 70, 90 20, 110 40),
 (160 30, 100 100, 180 100))');
```



*Largest Empty Circle within a set of lines.*

```
SELECT radius,
 ST_AsText(center) AS center,
 ST_AsText(nearest) AS nearest
FROM ST_LargestEmptyCircle(
 St_Collect(
 'MULTIPOINT ((70 50), (60 130), (130 150), (80 90))',
 'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))',
 'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'
)
);
```



*Largest Empty Circle within a set of points, constrained to lie in a polygon. The constraint polygon boundary must be included as an obstacle, as well as specified as the constraint for the circle center.*

**Ver también**

[ST\\_MinimumBoundingRadius](#)

#### 7.14.14 ST\_MinimumBoundingCircle

`ST_MinimumBoundingCircle` — Returns the smallest circle polygon that contains a geometry.

##### Synopsis

geometry **ST\_MinimumBoundingCircle**(geometry geomA, integer num\_segs\_per\_qt\_circ=48);

##### Descripción

Returns the smallest circle polygon that contains a geometry.



##### Note

The bounding circle is approximated by a polygon with a default of 48 segments per quarter circle. Because the polygon is an approximation of the minimum bounding circle, some points in the input geometry may not be contained within the polygon. The approximation can be improved by increasing the number of segments. For applications where an approximation is not suitable [ST\\_MinimumBoundingRadius](#) may be used.

Use with [ST\\_GeomCollFromText](#) to get the minimum bounding circle of a set of geometries.

To compute two points lying on the minimum circle (the "maximum diameter") use [ST\\_LongestLine](#).

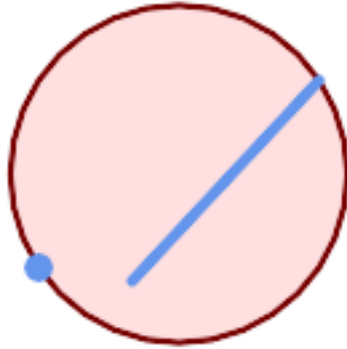
The ratio of the area of a polygon divided by the area of its Minimum Bounding Circle is referred to as the *Reock compactness score*.

Realizado por el módulo GEOS.

Disponibilidad: 1.4.0

## Ejemplos

```
SELECT d.disease_type,
 ST_MinimumBoundingCircle(ST_Collect(d.geom)) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```



*Minimum bounding circle of a point and linestring. Using 8 segs to approximate a quarter circle*

```
SELECT ST_AsText(ST_MinimumBoundingCircle(
 ST_Collect(
 ST_GeomFromText('LINESTRING(55 75,125 150)'),
 ST_Point(20, 80)), 8
)) As wktmbc;

wktmbc

POLYGON((135.59714732062 115,134.384753327498 102.690357210921,130.79416296937 ↵
 90.8537670908995,124.963360620072 79.9451031602111,117.116420743937 ↵
 70.3835792560632,107.554896839789 62.5366393799277,96.6462329091006 ↵
 56.70583703063,84.8096427890789 53.115246672502,72.5000000000001 ↵
 51.9028526793802,60.1903572109213 53.1152466725019,48.3537670908996 ↵
 56.7058370306299,37.4451031602112 62.5366393799276,27.8835792560632 ↵
 70.383579256063,20.0366393799278 79.9451031602109,14.20583703063 ↵
 90.8537670908993,10.615246672502 102.690357210921,9.40285267938019 115,10.6152466725019 ↵
 127.309642789079,14.2058370306299 139.1462329091,20.0366393799275 ↵
 150.054896839789,27.883579256063 159.616420743937,
 37.4451031602108 167.463360620072,48.3537670908992 173.29416296937,60.190357210921 ↵
 176.884753327498,
 72.4999999999998 178.09714732062,84.8096427890786 176.884753327498,96.6462329091003 ↵
 173.29416296937,107.554896839789 167.463360620072,
 117.116420743937 159.616420743937,124.963360620072 150.054896839789,130.79416296937 ↵
 139.146232909101,134.384753327498 127.309642789079,135.59714732062 115))
```

## Ver también

[ST\\_GeomCollFromText](#), [ST\\_MinimumBoundingRadius](#), [ST\\_LargestEmptyCircle](#), [ST\\_LongestLine](#)

### 7.14.15 ST\_MinimumBoundingRadius

**ST\_MinimumBoundingRadius** — Returns the center point and radius of the smallest circle that contains a geometry.

**Synopsis**

(geometry, double precision) **ST\_MinimumBoundingRadius**(geometry geom);

**Descripción**

Computes the center point and radius of the smallest circle that contains a geometry. Returns a record with fields:

- `center` - center point of the circle
- `radius` - radius of the circle

Use with **ST\_GeomCollFromText** to get the minimum bounding circle of a set of geometries.

To compute two points lying on the minimum circle (the "maximum diameter") use **ST\_LongestLine**.

Disponibilidad: 2.3.0

**Ejemplos**

```
SELECT ST_AsText(center), radius FROM ST_MinimumBoundingRadius('POLYGON((26426 65078,26531 65242,26075 65136,26096 65427,26426 65078))');
```

| st_astext                                | radius           |
|------------------------------------------|------------------|
| POINT(26284.8418027133 65267.1145090825) | 247.436045591407 |

**Ver también**

**ST\_GeomCollFromText**, **ST\_MinimumBoundingCircle**, **ST\_LongestLine**

**7.14.16 ST\_OrientedEnvelope**

**ST\_OrientedEnvelope** — Returns a minimum-area rectangle containing a geometry.

**Synopsis**

geometry **ST\_OrientedEnvelope**( geometry geom );

**Descripción**

Returns the minimum-area rotated rectangle enclosing a geometry. Note that more than one such rectangle may exist. May return a Point or LineString in the case of degenerate inputs.

Availability: 2.5.0.

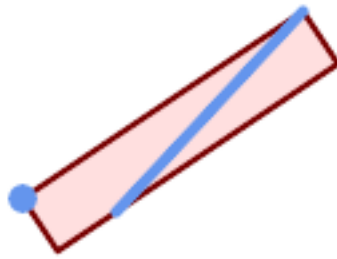
Requires GEOS >= 3.6.0.

## Ejemplos

```
SELECT ST_AsText(ST_OrientedEnvelope('MULTIPOINT ((0 0), (-1 -1), (3 2))'));

 st_astext

POLYGON((3 2,2.88 2.16,-1.12 -0.84,-1 -1,3 2))
```



*Oriented envelope of a point and linestring.*

```
SELECT ST_AsText(ST_OrientedEnvelope(
 ST_Collect(
 ST_GeomFromText('LINESTRING(55 75,125 150)'),
 ST_Point(20, 80))
)) As wktenv;

wktenv

POLYGON((19.9999999999997 79.9999999999999,33.0769230769229 ↔
 60.3846153846152,138.076923076924 130.384615384616,125.000000000001 ↔
 150.000000000001,19.9999999999997 79.9999999999999))
```

## Ver también

[ST\\_Envelope](#) [ST\\_MinimumBoundingCircle](#)

## 7.14.17 ST\_OffsetCurve

**ST\_OffsetCurve** — Returns an offset line at a given distance and side from an input line.

### Synopsis

geometry **ST\_OffsetCurve**(geometry line, float signed\_distance, text style\_parameters=’');



## Descripción

Return an offset line at a given distance and side from an input line. All points of the returned geometries are not further than the given distance from the input geometry. Useful for computing parallel lines about a center line.

For positive distance the offset is on the left side of the input line and retains the same direction. For a negative distance it is on the right side and in the opposite direction.

Units of distance are measured in units of the spatial reference system.

Note that output may be a MULTILINESTRING or EMPTY for some jigsaw-shaped input geometries.

The optional third parameter allows specifying a list of blank-separated key=value pairs to tweak operations as follows:

- 'quad\_segs=#' : number of segments used to approximate a quarter circle (defaults to 8).
- 'join=round|mitre|bevel' : join style (defaults to "round"). 'miter' is also accepted as a synonym for 'mitre'.
- 'mitre\_limit=#.#' : mitre ratio limit (only affects mitred join style). 'miter\_limit' is also accepted as a synonym for 'mitre\_limit'.

Realizado por el módulo GEOS.

Disponibilidad: 2.0

Enhanced: 2.5 - added support for GEOMETRYCOLLECTION and MULTILINESTRING



### Note

This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry.

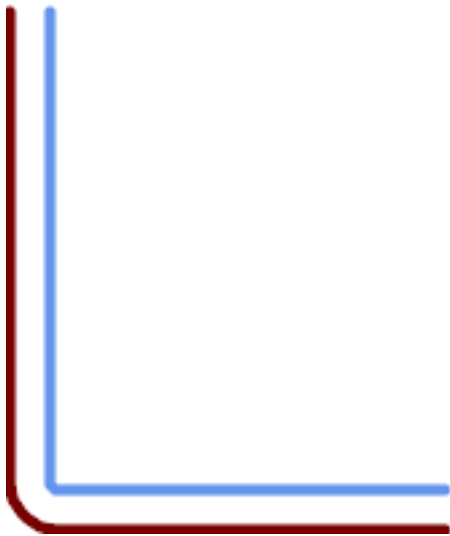
---

## Ejemplos

Compute an open buffer around roads

```
SELECT ST_Union(
 ST_OffsetCurve(f.geom, f.width/2, 'quad_segs=4 join=round'),
 ST_OffsetCurve(f.geom, -f.width/2, 'quad_segs=4 join=round')
) as track
FROM someroadstable;
```

---



*15, 'quad\_segs=4 join=round' original line and its offset 15 units.*

```
SELECT ST_AsText(ST_OffsetCurve(↵
 ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104 ↵
 16,84 16,64 16,
 44 16,24 16,20 16,18 16,17 17,
 16 18,16 20,16 40,16 60,16 80,16 100,
 16 120,16 140,16 160,16 180,16 195)') ↵
 ,
 15, 'quad_segs=4 join=round'));
```

output

```
LINESTRING(164 1,18 1,12.2597485145237 ↵
 2.1418070123307,
 7.39339828220179 5.39339828220179,
 5.39339828220179 7.39339828220179,
 2.14180701233067 12.2597485145237,1 ↵
 18,1 195)
```

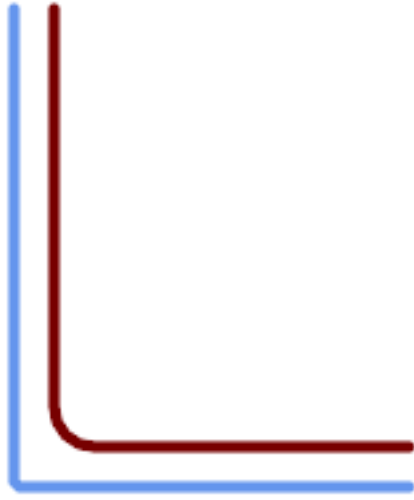


*-15, 'quad\_segs=4 join=round' original line and its offset -15 units*

```
SELECT ST_AsText(ST_OffsetCurve(geom,
 -15, 'quad_segs=4 join=round')) As ↵
 notsocurvy
FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104 ↵
 16,84 16,64 16,
 44 16,24 16,20 16,18 16,17 17,
 16 18,16 20,16 40,16 60,16 80,16 100,
 16 120,16 140,16 160,16 180,16 195)') ↵
 As geom;
```

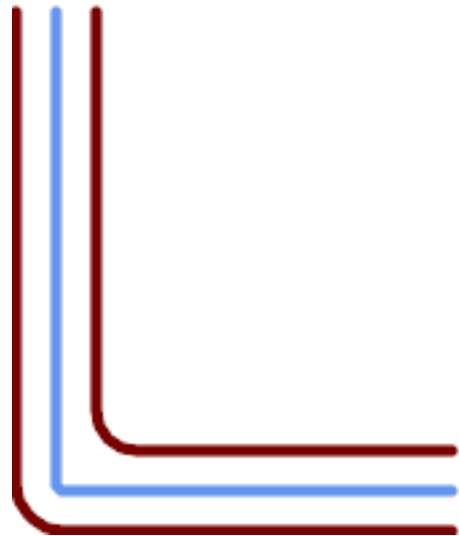
notsocurvy

```
LINESTRING(31 195,31 31,164 31)
```



*double-offset to get more curvy, note the first reverses direction, so  $-30 + 15 = -15$*

```
SELECT ST_AsText(ST_OffsetCurve(↵
 ST_OffsetCurve(geom,↵
 -30, 'quad_segs=4 join=round'), -15,↵
 'quad_segs=4 join=round')) As morecurvy
FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104↵
 16,84 16,64 16,↵
 44 16,24 16,20 16,18 16,17 17,↵
 16 18,16 20,16 40,16 60,16 80,16 100,↵
 16 120,16 140,16 160,16 180,16 195)')↵
 As geom;
morecurvy
LINESTRING(164 31,46 31,40.2597485145236↵
 32.1418070123307,↵
 35.3933982822018 35.3933982822018,↵
 32.1418070123307 40.2597485145237,31↵
 46,31 195)
```



*double-offset to get more curvy,combined with regular offset 15 to get parallel lines. Overlaid with original.*

```
SELECT ST_AsText(ST_Collect(↵
 ST_OffsetCurve(geom, 15, 'quad_segs=4↵
 join=round'),↵
 ST_OffsetCurve(ST_OffsetCurve(geom,↵
 -30, 'quad_segs=4 join=round'), -15,↵
 'quad_segs=4 join=round')↵
) As parallel_curves
FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104↵
 16,84 16,64 16,↵
 44 16,24 16,20 16,18 16,17 17,↵
 16 18,16 20,16 40,16 60,16 80,16 100,↵
 16 120,16 140,16 160,16 180,16 195)')↵
 As geom;
parallel curves
MULTILINESTRING((164 1,18↵
 1,12.2597485145237 2.1418070123307,↵
 7.39339828220179↵
 5.39339828220179,5.39339828220179 7.39339828220179,↵
 2.14180701233067 12.2597485145237,1 18,1↵
 195),↵
 (164 31,46 31,40.2597485145236↵
 32.1418070123307,35.3933982822018 35.3933982822018,↵
 32.1418070123307 40.2597485145237,31↵
 46,31 195))
```

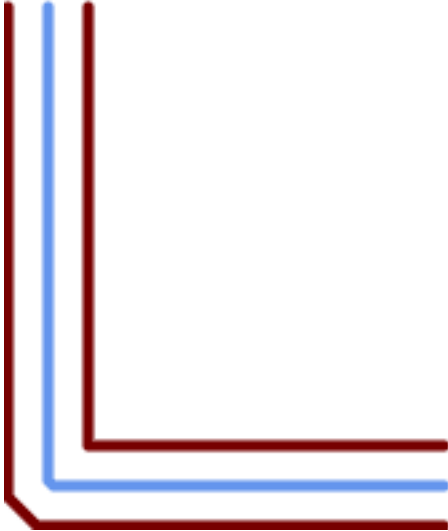


*15, 'quad\_segs=4 join=bevel' shown with original line*

```
SELECT ST_AsText(ST_OffsetCurve(↵
 ST_GeomFromText(↵
'LINESTRING(164 16,144 16,124 16,104 ↵
 16,84 16,64 16,↵
 44 16,24 16,20 16,18 16,17 17,↵
 16 18,16 20,16 40,16 60,16 80,16 100,↵
 16 120,16 140,16 160,16 180,16 195)') ↵
 ' 15, 'quad_segs=4 join=bevel'));
```

output

```
LINESTRING(164 1,18 1,7.39339828220179 ↵
 5.39339828220179,↵
 5.39339828220179 7.39339828220179,1 ↵
 18,1 195)
```



*15,-15 collected, join=mitre mitre\_limit=2.1*

```
SELECT ST_AsText(ST_Collect(↵
 ST_OffsetCurve(geom, 15, 'quad_segs=4 ↵
 join=mitre mitre_limit=2.2'),↵
 ST_OffsetCurve(geom, -15, 'quad_segs ↵
 =4 join=mitre mitre_limit=2.2')↵
))↵
FROM ST_GeomFromText(↵
'LINESTRING(164 16,144 16,124 16,104 ↵
 16,84 16,64 16,↵
 44 16,24 16,20 16,18 16,17 17,↵
 16 18,16 20,16 40,16 60,16 80,16 100,↵
 16 120,16 140,16 160,16 180,16 195)') ↵
 As geom;
```

output

```
MULTILINESTRING((164 1,11.7867965644036 ↵
 1,1 11.7867965644036,1 195),↵
 (31 195,31 31,164 31))
```

Ver también

[ST\\_Buffer](#)

7.14.18 ST\_PointOnSurface

ST\_PointOnSurface — Computes a point guaranteed to lie in a polygon, or on a geometry.

Synopsis

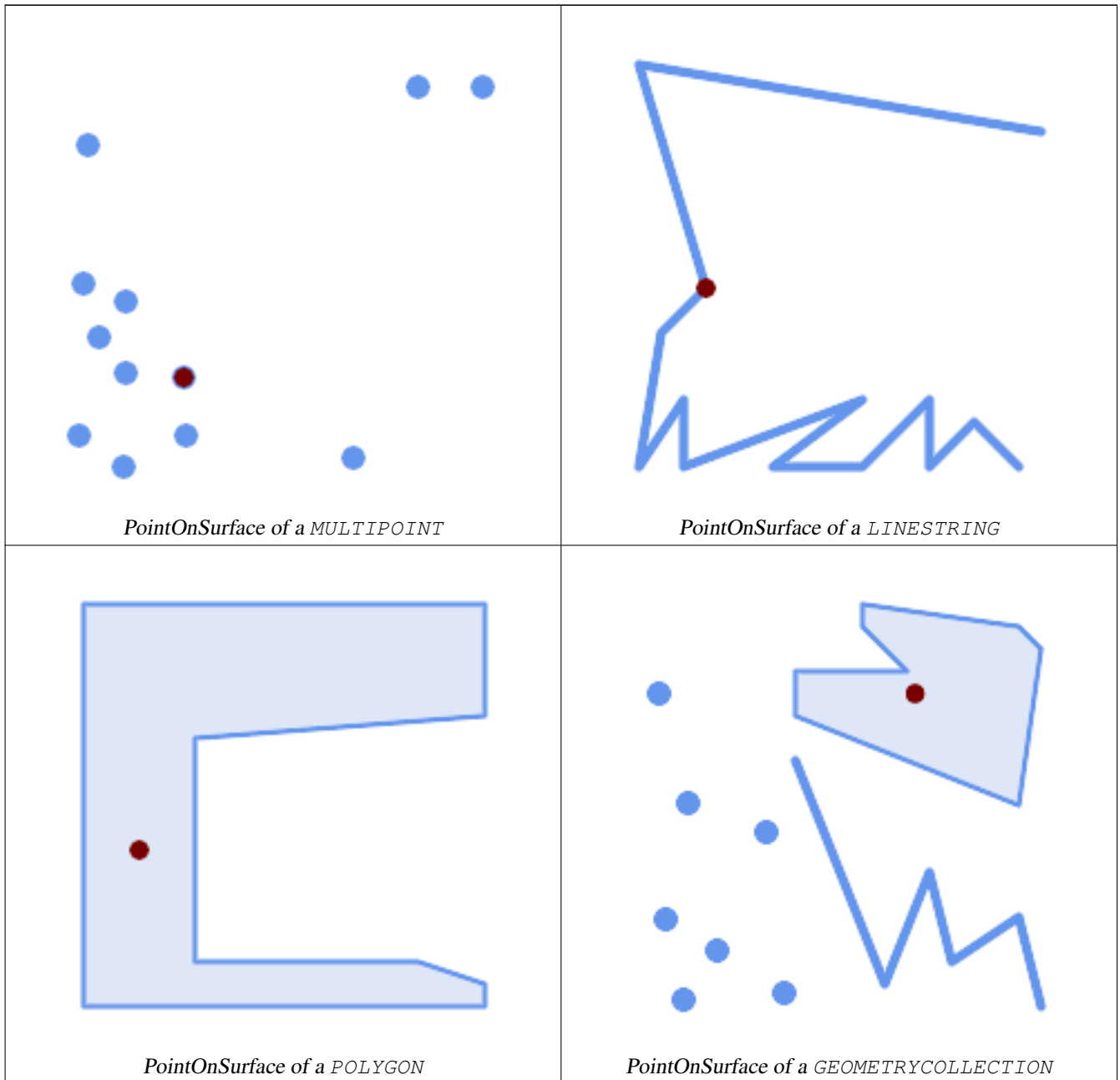
geometry **ST\_PointOnSurface**(geometry g1);

Descripción

Returns a POINT which is guaranteed to lie in the interior of a surface (POLYGON, MULTIPOLYGON, and CURVED POLYGON). In PostGIS this function also works on line and point geometries.

- ✓ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.14.2 // s3.2.18.2
- ✓ This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. The specifications define ST\_PointOnSurface for surface geometries only. PostGIS extends the function to support all common geometry types. Other databases (Oracle, DB2, ArcSDE) seem to support this function only for surfaces. SQL Server 2008 supports all common geometry types.
- ✓ This function supports 3d and will not drop the z-index.

## Ejemplos



```
SELECT ST_AsText(ST_PointOnSurface('POINT(0 5)::geometry'));

POINT(0 5)
```

```
SELECT ST_AsText(ST_PointOnSurface('LINESTRING(0 5, 0 10)::geometry));

POINT(0 5)

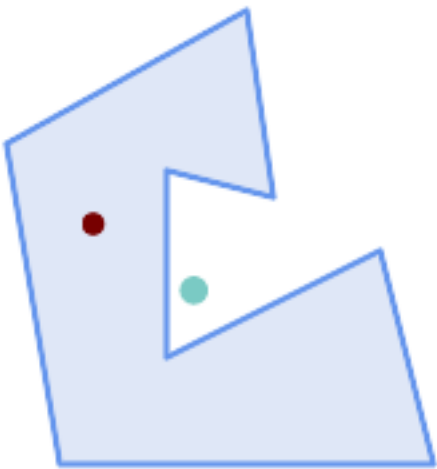
SELECT ST_AsText(ST_PointOnSurface('POLYGON((0 0, 0 5, 5 5, 5 0, 0 0))::geometry));

POINT(2.5 2.5)

SELECT ST_AsEWKT(ST_PointOnSurface(ST_GeomFromEWKT('LINESTRING(0 5 1, 0 0 1, 0 10 2)')));

POINT(0 0 1)
```

**Example:** The result of ST\_PointOnSurface is guaranteed to lie within polygons, whereas the point computed by **ST\_Centroid** may be outside.



*Red: point on surface; Green: centroid*

```
SELECT ST_AsText(ST_PointOnSurface(geom)) AS pt_on_surf,
 ST_AsText(ST_Centroid(geom)) AS centroid
FROM (SELECT 'POLYGON ((130 120, 120 190, 30 140, 50 20, 190 20,
 170 100, 90 60, 90 130, 130 120))::geometry AS geom) AS t;

pt_on_surf | centroid
-----+-----
POINT(62.5 110) | POINT(100.18264840182648 85.11415525114155)
```

**Ver también**

**ST\_Centroid**, **ST\_MaximumInscribedCircle**

**7.14.19 ST\_Polygonize**

ST\_Polygonize — Computes a collection of polygons formed from the linework of a set of geometries.

**Synopsis**

```
geometry ST_Polygonize(geometry set geomfield);
geometry ST_Polygonize(geometry[] geom_array);
```

## Descripción

Creates a GeometryCollection containing the polygons formed by the constituent linework of a set of geometries. Input linework must be correctly noded for this function to work properly.



### Note

To ensure input is fully noded use [ST\\_Node](#) on the input geometry before polygonizing.



### Note

GeometryCollections are often difficult to deal with with third party tools. Use [ST\\_Dump](#) to convert the polygonize result into separate polygons.

Realizado por el módulo GEOS.

Disponibilidad: 1.0.0RC1

## Examples: Polygonizing single linestrings

```
SELECT ST_AsEWKT(ST_Polygonize(geom_4269)) As geomtextrep
FROM (SELECT geom_4269 FROM ma.suffolk_edges ORDER BY tlid LIMIT 45) As foo;
```

geomtextrep  
-----

```
SRID=4269;GEOMETRYCOLLECTION(POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 42.285752,-71.040878 42.285678)),
POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,-71.171794 42.354971,-71.170511 42.354855,
-71.17112 42.354238,-71.17166 42.353675)))
(1 row)
```

--Use ST\_Dump to dump out the polygonize geoms into individual polygons

```
SELECT ST_AsEWKT((ST_Dump(foofoo.polycoll)).geom) As geomtextrep
FROM (SELECT ST_Polygonize(geom_4269) As polycoll
 FROM (SELECT geom_4269 FROM ma.suffolk_edges
 ORDER BY tlid LIMIT 45) As foo) As foofoo;
```

geomtextrep  
-----

```
SRID=4269;POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 42.285752,-71.040878 42.285678))
SRID=4269;POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,-71.171794 42.354971,-71.170511 42.354855,-71.17112 42.354238,-71.17166 42.353675))
(2 rows)
```

## Ver también

[ST\\_Node](#), [ST\\_Dump](#)

## 7.14.20 ST\_ReducePrecision

**ST\_ReducePrecision** — Returns a valid geometry with points rounded to a grid tolerance.

## Synopsis

geometry **ST\_ReducePrecision**(geometry g, float8 gridsz);

## Descripción

Returns a valid geometry with all points rounded to the provided grid tolerance, and features below the tolerance removed.

Unlike **ST\_SnapToGrid** the returned geometry will be valid, with no ring self-intersections or collapsed components.

Precision reduction can be used to:

- match coordinate precision to the data accuracy
- reduce the number of coordinates needed to represent a geometry
- ensure valid geometry output to formats which use lower precision (e.g. text formats such as WKT, GeoJSON or KML when the number of output decimal places is limited).
- export valid geometry to systems which use lower or limited precision (e.g. SDE, Oracle tolerance value)

Availability: 3.1.0.

Requires GEOS >= 3.9.0.

## Ejemplos

```
SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 0.1));
 st_astext

POINT(1.4 19.3)

SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 1.0));
 st_astext

POINT(1 19)

SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 10));
 st_astext

POINT(0 20)
```

### Precision reduction can reduce number of vertices

```
SELECT ST_AsText(ST_ReducePrecision('LINESTRING (10 10, 19.6 30.1, 20 30, 20.3 30, 40 40)', ↵
 1));
 st_astext

LINESTRING (10 10, 20 30, 40 40)
```

### Precision reduction splits polygons if needed to ensure validity

```
SELECT ST_AsText(ST_ReducePrecision('POLYGON ((10 10, 60 60.1, 70 30, 40 40, 50 10, 10 10)) ↵
 ', 10));
 st_astext

MULTIPOLYGON (((60 60, 70 30, 40 40, 60 60)), ((40 40, 50 10, 10 10, 40 40)))
```



**Ver también**

[ST\\_SnapToGrid](#), [ST\\_Simplify](#), [ST\\_SimplifyVW](#)

### 7.14.21 ST\_SharedPaths

**ST\_SharedPaths** — Returns a collection containing paths shared by the two input linestrings/multilinestrings.

#### Synopsis

```
geometry ST_SharedPaths(geometry lineal1, geometry lineal2);
```

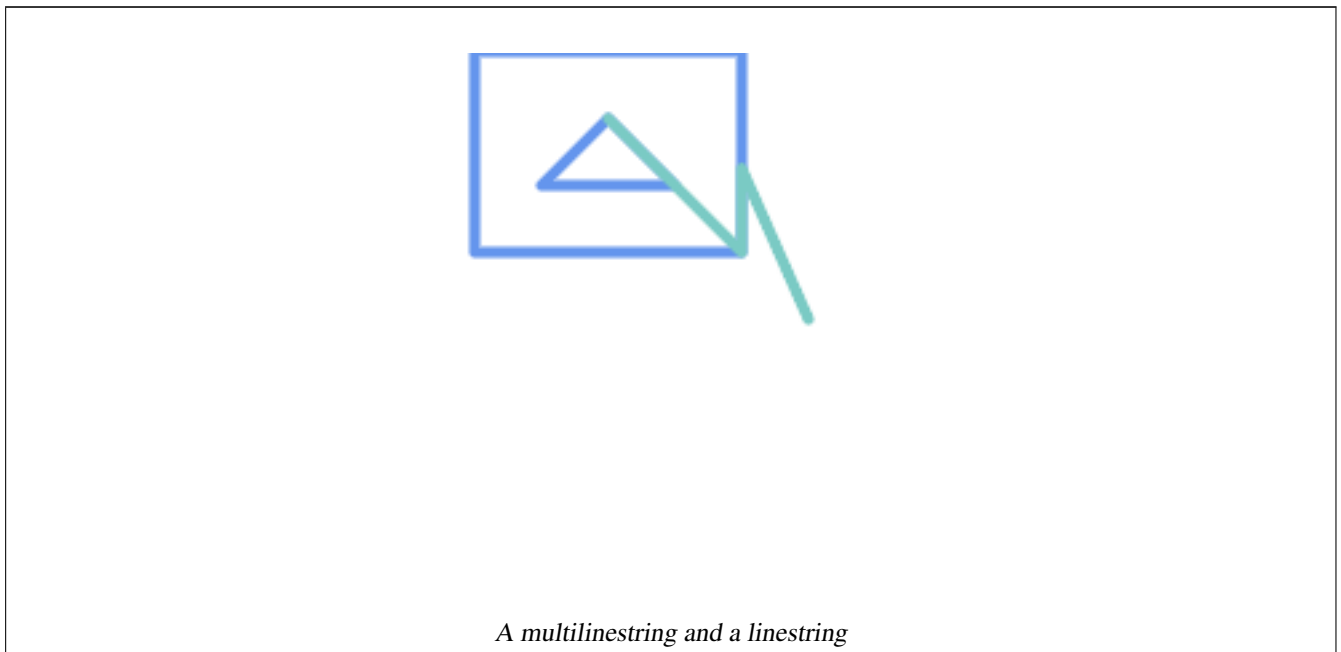
#### Descripción

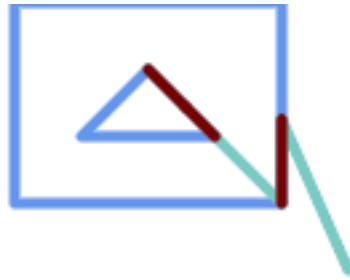
Returns a collection containing paths shared by the two input geometries. Those going in the same direction are in the first element of the collection, those going in the opposite direction are in the second element. The paths themselves are given in the direction of the first geometry.

Realizado por el módulo GEOS.

Disponibilidad: 2.0.0

#### Examples: Finding shared paths





*The shared path of multilinestring and linestring overlaid with original geometries.*

```
SELECT ST_AsText(
 ST_SharedPaths(
 ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
 (51 150,101 150,76 175,51 150))'),
 ST_GeomFromText('LINESTRING(151 100,126 156.25,126 125,90 161, 76 175)')
) As wkt
```

wkt

```

GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125),
 (101 150,90 161),(90 161,76 175)),MULTILINESTRING EMPTY)
```

same example but linestring orientation flipped

```
SELECT ST_AsText(
 ST_SharedPaths(
 ST_GeomFromText('LINESTRING(76 175,90 161,126 125,126 156.25,151 100)'),
 ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
 (51 150,101 150,76 175,51 150))')
) As wkt
```

wkt

```

GEOMETRYCOLLECTION(MULTILINESTRING EMPTY,
 MULTILINESTRING((76 175,90 161),(90 161,101 150),(126 125,126 156.25)))
```

**Ver también**

[ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

### 7.14.22 ST\_Simplify

ST\_Simplify — Returns a simplified version of a geometry, using the Douglas-Peucker algorithm.

#### Synopsis

geometry **ST\_Simplify**(geometry geomA, float tolerance);  
geometry **ST\_Simplify**(geometry geomA, float tolerance, boolean preserveCollapsed);

#### Descripción

Returns a "simplified" version of the given geometry using the Douglas-Peucker algorithm. Will actually do something only with (multi)lines and (multi)polygons but you can safely call it with any kind of geometry. Since simplification occurs on a object-by-object basis you can also feed a GeometryCollection to this function.

The "preserve collapsed" flag will retain objects that would otherwise be too small given the tolerance. For example, a 1m long line simplified with a 10m tolerance. If `preserveCollapsed` argument is specified as true, the line will not disappear. This flag is useful for rendering engines, to avoid having large numbers of very small objects disappear from a map leaving surprising gaps.



**Note**  
Note that returned geometry might lose its simplicity (see [ST\\_IsSimple](#))



**Note**  
Note topology may not be preserved and may result in invalid geometries. Use (see [ST\\_SimplifyPreserveTopology](#)) to preserve topology.

Disponibilidad: 1.2.2

#### Ejemplos

A circle simplified too much becomes a triangle, medium an octagon,

```
SELECT ST_Npoints(geom) AS np_before,
 ST_NPoints(ST_Simplify(geom,0.1)) AS np01_notbadcircle,
 ST_NPoints(ST_Simplify(geom,0.5)) AS np05_notquitecircle,
 ST_NPoints(ST_Simplify(geom,1)) AS np1_octagon,
 ST_NPoints(ST_Simplify(geom,10)) AS np10_triangle,
 (ST_Simplify(geom,100) is null) AS np100_geometrygoesaway
FROM
 (SELECT ST_Buffer('POINT(1 3)', 10,12) As geom) AS foo;
```

| np_before | np01_notbadcircle | np05_notquitecircle | np1_octagon | np10_triangle | np100_geometrygoesaway |
|-----------|-------------------|---------------------|-------------|---------------|------------------------|
| 49        | 33                | 17                  | 9           | 4             | t                      |

#### Ver también

[ST\\_IsSimple](#), [ST\\_SimplifyPreserveTopology](#), [ST\\_SimplifyVW](#), [Topology ST\\_Simplify](#)



## Descripción

Computes a simplified topology-preserving outer or inner hull of a polygonal geometry. An outer hull completely covers the input geometry. An inner hull is completely covered by the input geometry. The result is a polygonal geometry formed by a subset of the input vertices. MultiPolygons and holes are handled and produce a result with the same structure as the input.

The reduction in vertex count is controlled by the `vertex_fraction` parameter, which is a number in the range 0 to 1. Lower values produce simpler results, with smaller vertex count and less concaveness. For both outer and inner hulls a vertex fraction of 1.0 produces the original geometry. For outer hulls a value of 0.0 produces the convex hull (for a single polygon); for inner hulls it produces a triangle.

The simplification process operates by progressively removing concave corners that contain the least amount of area, until the vertex count target is reached. It prevents edges from crossing, so the result is always a valid polygonal geometry.

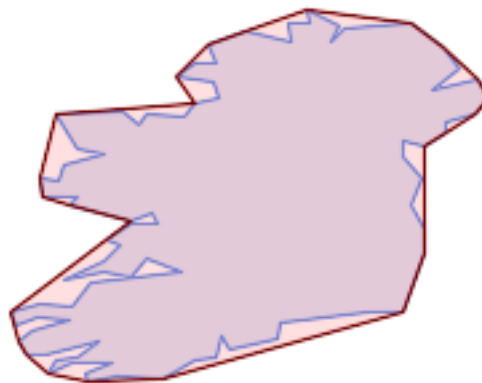
To get better results with geometries that contain relatively long line segments, it might be necessary to "segmentize" the input, as shown below.

Realizado por el módulo GEOS.

Availability: 3.3.0.

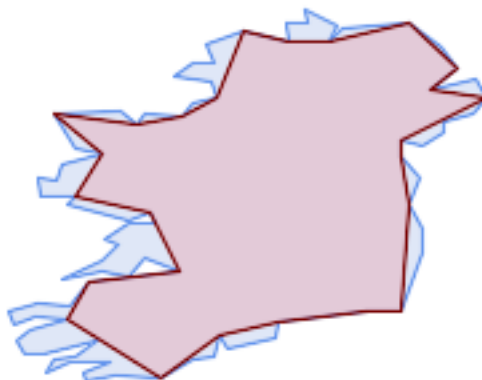
Requires GEOS >= 3.11.0.

## Ejemplos



*Outer hull of a Polygon*

```
SELECT ST_SimplifyPolygonHull(
 'POLYGON ((131 158, 136 163, 161 165, 173 156, 179 148, 169 140, 186 144, 190 137, 185 ↵
 131, 174 128, 174 124, 166 119, 158 121, 158 115, 165 107, 161 97, 166 88, 166 79, 158 ↵
 57, 145 57, 112 53, 111 47, 93 43, 90 48, 88 40, 80 39, 68 32, 51 33, 40 31, 39 34, ↵
 49 38, 34 38, 25 34, 28 39, 36 40, 44 46, 24 41, 17 41, 14 46, 19 50, 33 54, 21 55, 13 ↵
 52, 11 57, 22 60, 34 59, 41 68, 75 72, 62 77, 56 70, 46 72, 31 69, 46 76, 52 82, 47 ↵
 84, 56 90, 66 90, 64 94, 56 91, 33 97, 36 100, 23 100, 22 107, 29 106, 31 112, 46 116, ↵
 36 118, 28 131, 53 132, 59 127, 62 131, 76 130, 80 135, 89 137, 87 143, 73 145, 80 ↵
 150, 88 150, 85 157, 99 162, 116 158, 115 165, 123 165, 122 170, 134 164, 131 158))',
 0.3);
```



*Inner hull of a Polygon*

```
SELECT ST_SimplifyPolygonHull(
 'POLYGON ((131 158, 136 163, 161 165, 173 156, 179 148, 169 140, 186 144, 190 137, 185 ↵
 131, 174 128, 174 124, 166 119, 158 121, 158 115, 165 107, 161 97, 166 88, 166 79, 158 ↵
 57, 145 57, 112 53, 111 47, 93 43, 90 48, 88 40, 80 39, 68 32, 51 33, 40 31, 39 34, ↵
 49 38, 34 38, 25 34, 28 39, 36 40, 44 46, 24 41, 17 41, 14 46, 19 50, 33 54, 21 55, 13 ↵
 52, 11 57, 22 60, 34 59, 41 68, 75 72, 62 77, 56 70, 46 72, 31 69, 46 76, 52 82, 47 ↵
 84, 56 90, 66 90, 64 94, 56 91, 33 97, 36 100, 23 100, 22 107, 29 106, 31 112, 46 116, ↵
 36 118, 28 131, 53 132, 59 127, 62 131, 76 130, 80 135, 89 137, 87 143, 73 145, 80 ↵
 150, 88 150, 85 157, 99 162, 116 158, 115 165, 123 165, 122 170, 134 164, 131 158))',
 0.3, false);
```



*Outer hull simplification of a MultiPolygon, with segmentization*

```
SELECT ST_SimplifyPolygonHull(
 ST_Segmentize(ST_Letters('xt'), 2.0),
 0.1);
```

#### Ver también

[ST\\_ConvexHull](#), [ST\\_SimplifyVW](#), [ST\\_ConcaveHull](#), [ST\\_Segmentize](#)

### 7.14.25 ST\_SimplifyVW

ST\_SimplifyVW — Returns a simplified version of a geometry, using the Visvalingam-Whyatt algorithm

#### Synopsis

geometry **ST\_SimplifyVW**(geometry geomA, float tolerance);

#### Descripción

Returns a "simplified" version of the given geometry using the Visvalingam-Whyatt algorithm. Will actually do something only with (multi)lines and (multi)polygons but you can safely call it with any kind of geometry. Since simplification occurs on a object-by-object basis you can also feed a GeometryCollection to this function.



#### Note

Note that returned geometry might lose its simplicity (see [ST\\_IsSimple](#))



#### Note

Note topology may not be preserved and may result in invalid geometries. Use (see [ST\\_SimplifyPreserveTopology](#)) to preserve topology.



#### Note

This function handles 3D and the third dimension will affect the result.

Disponibilidad: 2.2.0

#### Ejemplos

A LineString is simplified with a minimum area threshold of 30.

```
select ST_AsText(ST_SimplifyVW(geom,30)) simplified
FROM (SELECT 'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry geom) As foo;
-result
simplified

LINESTRING(5 2,7 25,10 10)
```

#### Ver también

[ST\\_SetEffectiveArea](#), [ST\\_Simplify](#), [ST\\_SimplifyPreserveTopology](#), [Topology ST\\_Simplify](#)

### 7.14.26 ST\_SetEffectiveArea

ST\_SetEffectiveArea — Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.

## Synopsis

geometry **ST\_SetEffectiveArea**(geometry geomA, float threshold = 0, integer set\_area = 1);

## Descripción

Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm. The effective area is stored as the M-value of the vertex. If the optional "theshold" parameter is used, a simplified geometry will be returned, containing only vertices with an effective area greater than or equal to the threshold value.

This function can be used for server-side simplification when a threshold is specified. Another option is to use a threshold value of zero. In this case, the full geometry will be returned with effective areas as M-values, which can be used by the client to simplify very quickly.

Will actually do something only with (multi)lines and (multi)polygons but you can safely call it with any kind of geometry. Since simplification occurs on a object-by-object basis you can also feed a GeometryCollection to this function.



### Note

Note that returned geometry might lose its simplicity (see [ST\\_IsSimple](#))



### Note

Note topology may not be preserved and may result in invalid geometries. Use (see [ST\\_SimplifyPreserveTopology](#)) to preserve topology.



### Note

The output geometry will lose all previous information in the M-values



### Note

This function handles 3D and the third dimension will affect the effective area

Disponibilidad: 2.2.0

## Ejemplos

Calculating the effective area of a LineString. Because we use a threshold value of zero, all vertices in the input geometry are returned.

```
select ST_AsText(ST_SetEffectiveArea(geom)) all_pts, ST_AsText(ST_SetEffectiveArea(geom,30) ←
) thrshld_30
FROM (SELECT 'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry geom) As foo;
--result
all_pts | thrshld_30
-----+-----+
LINESTRING M (5 2 3.40282346638529e+38,3 8 29,6 20 1.5,7 25 49.5,10 10 3.40282346638529e ←
+38) | LINESTRING M (5 2 3.40282346638529e+38,7 25 49.5,10 10 3.40282346638529e+38)
```



Ver también

[ST\\_SimplifyVW](#)

## 7.14.27 ST\_TriangulatePolygon

ST\_TriangulatePolygon — Computes the constrained Delaunay triangulation of polygons

### Synopsis

geometry **ST\_TriangulatePolygon**(geometry geom);

### Descripción

Computes the constrained Delaunay triangulation of polygons. Holes and Multipolygons are supported.

The "constrained Delaunay triangulation" of a polygon is a set of triangles formed from the vertices of the polygon, and covering it exactly, with the maximum total interior angle over all possible triangulations. It provides the "best quality" triangulation of the polygon.

Availability: 3.3.0.

Requires GEOS >= 3.11.0.

### Example

Triangulation of a square.

```
SELECT ST_AsText(
 ST_TriangulatePolygon('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'));

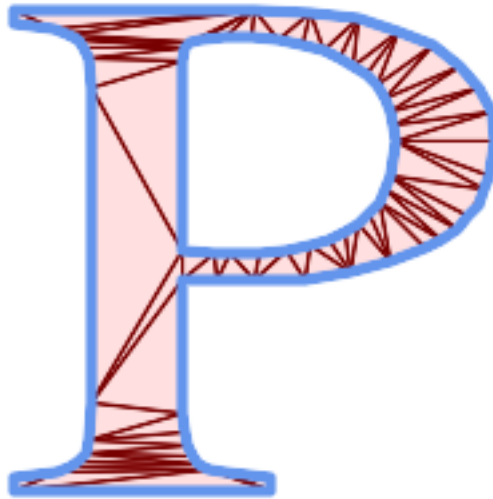
 st_astext

GEOMETRYCOLLECTION(POLYGON((0 0,0 1,1 1,0 0)),POLYGON((1 1,1 0,0 0,1 1)))
```

### Example

Triangulation of the letter P.

```
SELECT ST_AsText(ST_TriangulatePolygon(
 'POLYGON ((26 17, 31 19, 34 21, 37 24, 38 29, 39 43, 39 161, 38 172, 36 176, 34 179, 30 ↵
 181, 25 183, 10 185, 10 190, 100 190, 121 189, 139 187, 154 182, 167 177, 177 169, ↵
 184 161, 189 152, 190 141, 188 128, 186 123, 184 117, 180 113, 176 108, 170 104, 164 ↵
 101, 151 96, 136 92, 119 89, 100 89, 86 89, 73 89, 73 39, 74 32, 75 27, 77 23, 79 ↵
 20, 83 18, 89 17, 106 15, 106 10, 10 10, 10 15, 26 17), (152 147, 151 152, 149 157, ↵
 146 162, 142 166, 137 169, 132 172, 126 175, 118 177, 109 179, 99 180, 89 180, 80 ↵
 179, 76 178, 74 176, 73 171, 73 100, 85 99, 91 99, 102 99, 112 100, 121 102, 128 ↵
 104, 134 107, 139 110, 143 114, 147 118, 149 123, 151 128, 153 141, 152 147)))'
));
```



*Polygon Triangulation*

#### Ver también

[ST\\_DelaunayTriangles](#), [ST\\_ConstrainedDelaunayTriangles](#), [ST\\_Tessellate](#)

### 7.14.28 ST\_VoronoiLines

`ST_VoronoiLines` — Returns the boundaries of the Voronoi diagram of the vertices of a geometry.

#### Synopsis

geometry **ST\_VoronoiLines**( geometry geom , float8 tolerance = 0.0 , geometry extend\_to = NULL );

#### Descripción

Computes a two-dimensional **Voronoi diagram** from the vertices of the supplied geometry and returns the boundaries between cells in the diagram as a MultiLineString. Returns null if input geometry is null. Returns an empty geometry collection if the input geometry contains only one vertex. Returns an empty geometry collection if the `extend_to` envelope has zero area.

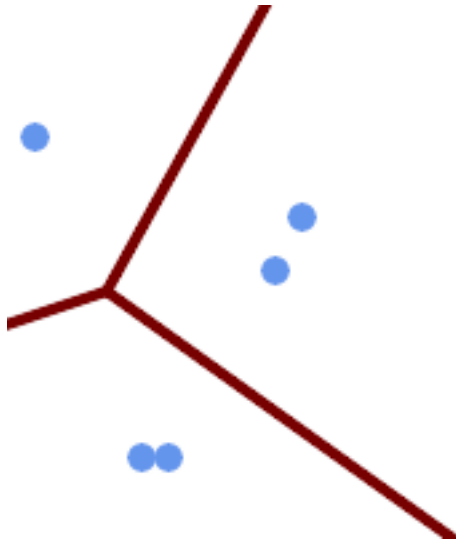
Optional parameters:

- `tolerance`: The distance within which vertices will be considered equivalent. Robustness of the algorithm can be improved by supplying a nonzero tolerance distance. (default = 0.0)
- `extend_to`: If present, the diagram is extended to cover the envelope of the supplied geometry, unless smaller than the default envelope (default = NULL, default envelope is the bounding box of the input expanded by about 50%).

Realizado por el módulo GEOS.

Disponibilidad: 2.3.0

## Ejemplos



*Voronoi diagram lines, with tolerance of 30 units*

```
SELECT ST_VoronoiLines(
 'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)::geometry,
 30) AS geom;
```

```
ST_AsText output
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273),(36.8181818181818 92.2727272727273, ←
 92.2727272727273,-110 43.3333333333333),(230 -45.7142857142858,36.8181818181818 92.2727272727273) ←
 92.2727272727273))
```

## Ver también

[ST\\_DelaunayTriangles](#), [ST\\_VoronoiPolygons](#)

### 7.14.29 ST\_VoronoiPolygons

**ST\_VoronoiPolygons** — Returns the cells of the Voronoi diagram of the vertices of a geometry.

#### Synopsis

```
geometry ST_VoronoiPolygons(geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL);
```

#### Descripción

Computes a two-dimensional **Voronoi diagram** from the vertices of the supplied geometry. The result is a **GEOMETRYCOLLECTION** of **POLYGONS** that covers an envelope larger than the extent of the input vertices. Returns null if input geometry is null. Returns an empty geometry collection if the input geometry contains only one vertex. Returns an empty geometry collection if the **extend\_to** envelope has zero area.

Optional parameters:

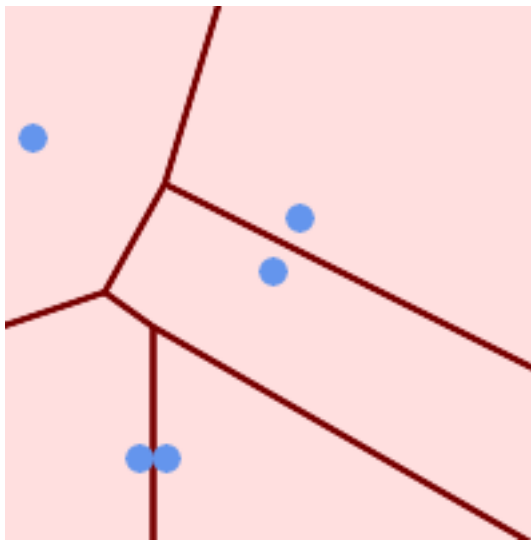
- **tolerance**: The distance within which vertices will be considered equivalent. Robustness of the algorithm can be improved by supplying a nonzero tolerance distance. (default = 0.0)

- `extend_to`: If present, the diagram is extended to cover the envelope of the supplied geometry, unless smaller than the default envelope (default = NULL, default envelope is the bounding box of the input expanded by about 50%).

Realizado por el módulo GEOS.

Disponibilidad: 2.3.0

## Ejemplos

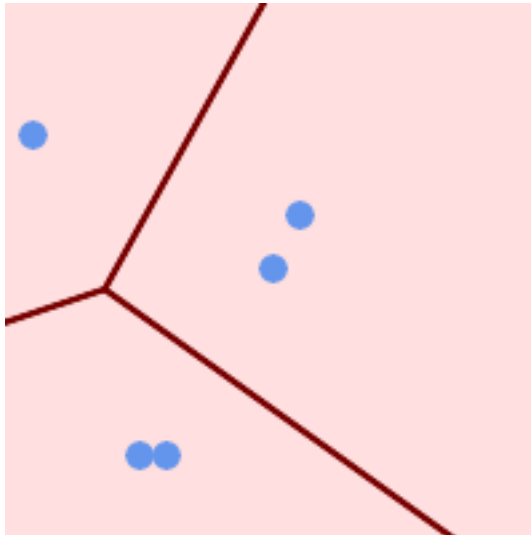


*Points overlaid on top of Voronoi diagram*

```
SELECT ST_VoronoiPolygons(
 'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120) '::geometry
) AS geom;
```

ST\_AsText output

```
GEOMETRYCOLLECTION(POLYGON((-110 43.3333333333333,-110 270,100.5 270,59.3478260869565 ↵
 132.826086956522,36.8181818181818 92.2727272727273,-110 43.3333333333333)),
POLYGON((55 -90,-110 -90,-110 43.3333333333333,36.8181818181818 92.2727272727273,55 ↵
 79.2857142857143,55 -90)),
POLYGON((230 47.5,230 -20.7142857142857,55 79.2857142857143,36.8181818181818 ↵
 92.2727272727273,59.3478260869565 132.826086956522,230 47.5)),POLYGON((230 ↵
 -20.7142857142857,230 -90,55 -90,55 79.2857142857143,230 -20.7142857142857)),
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```



*Voronoi diagram, with tolerance of 30 units*

```
SELECT ST_VoronoiPolygons(
 'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120) '::geometry,
 30) AS geom;
```

ST\_AsText output

```
GEOMETRYCOLLECTION(POLYGON((-110 43.3333333333333,-110 270,100.5 270,59.3478260869565 ↵
 132.826086956522,36.8181818181818 92.2727272727273,-110 43.3333333333333)),
POLYGON((230 47.5,230 -45.7142857142858,36.8181818181818 92.2727272727273,59.3478260869565 ↵
 132.826086956522,230 47.5)),POLYGON((230 -45.7142857142858,230 -90,-110 -90,-110 ↵
 43.3333333333333,36.8181818181818 92.2727272727273,230 -45.7142857142858)),
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```

**Ver también**

[ST\\_DelaunayTriangles](#), [ST\\_VoronoiLines](#)

## 7.15 Coverages

### 7.15.1 ST\_CoverageInvalidEdges

**ST\_CoverageInvalidEdges** — Window function that finds locations where polygons fail to form a valid coverage.

#### Synopsis

geometry **ST\_CoverageInvalidEdges**(geometry winset geom, float8 tolerance = 0);

#### Description

A window function which checks if the polygons in the window partition form a valid polygonal coverage. It returns linear indicators showing the location of invalid edges (if any) in each polygon.

A set of valid polygons is a valid coverage if the following conditions hold:

- **Non-overlapping** - polygons do not overlap (their interiors do not intersect)

- **Edge-Matched** - vertices along shared edges are identical

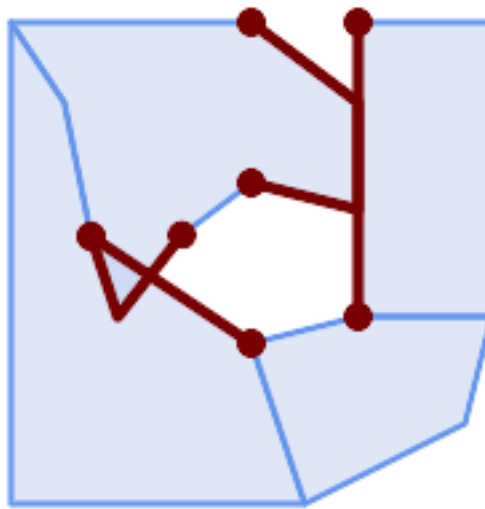
As a window function a value is returned for every input polygon. For polygons which violate one or more of the validity conditions the return value is a MULTILINESTRING containing the problematic edges. Coverage-valid polygons return the value NULL. Non-polygonal or empty geometries also produce NULL values.

The conditions allow a valid coverage to contain holes (gaps between polygons), as long as the surrounding polygons are edge-matched. However, very narrow gaps are often undesirable. If the *tolerance* parameter is specified with a non-zero distance, edges forming narrower gaps will also be returned as invalid.

The polygons being checked for coverage validity must also be valid geometries. This can be checked with [ST\\_IsValid](#).

Availability: 3.4.0 - requires GEOS >= 3.12.0

## Examples



*Invalid edges caused by overlap and non-matching vertices*

```
WITH coverage(id, geom) AS (VALUES
 (1, 'POLYGON ((10 190, 30 160, 40 110, 100 70, 120 10, 10 10, 10 190))'::geometry),
 (2, 'POLYGON ((100 190, 10 190, 30 160, 40 110, 50 80, 74 110.5, 100 130, 140 120, 140 160, 100 190))'::geometry),
 (3, 'POLYGON ((140 190, 190 190, 190 80, 140 80, 140 190))'::geometry),
 (4, 'POLYGON ((180 40, 120 10, 100 70, 140 80, 190 80, 180 40))'::geometry)
)
SELECT id, ST_AsText(ST_CoverageInvalidEdges(geom) OVER ())
FROM coverage;
```

| id | st_astext                                                                         |
|----|-----------------------------------------------------------------------------------|
| 1  | LINESTRING (40 110, 100 70)                                                       |
| 2  | MULTILINESTRING ((100 130, 140 120, 140 160, 100 190), (40 110, 50 80, 74 110.5)) |
| 3  | LINESTRING (140 80, 140 190)                                                      |
| 3  | null                                                                              |

```
-- Test entire table for coverage validity
SELECT true = ALL (
 SELECT ST_CoverageInvalidEdges(geom) OVER () IS NULL
 FROM coverage
);
```

**See Also**

[ST\\_IsValid](#), [ST\\_CoverageUnion](#), [ST\\_CoverageSimplify](#)

**7.15.2 ST\_CoverageSimplify**

`ST_CoverageSimplify` — Window function that simplifies the edges of a polygonal coverage.

**Synopsis**

geometry **ST\_CoverageSimplify**(geometry winset geom, float8 tolerance, boolean simplifyBoundary = true);

**Description**

A window function which simplifies the edges of polygons in a polygonal coverage. The simplification preserves the coverage topology. This means the simplified output polygons are consistent along shared edges, and still form a valid coverage.

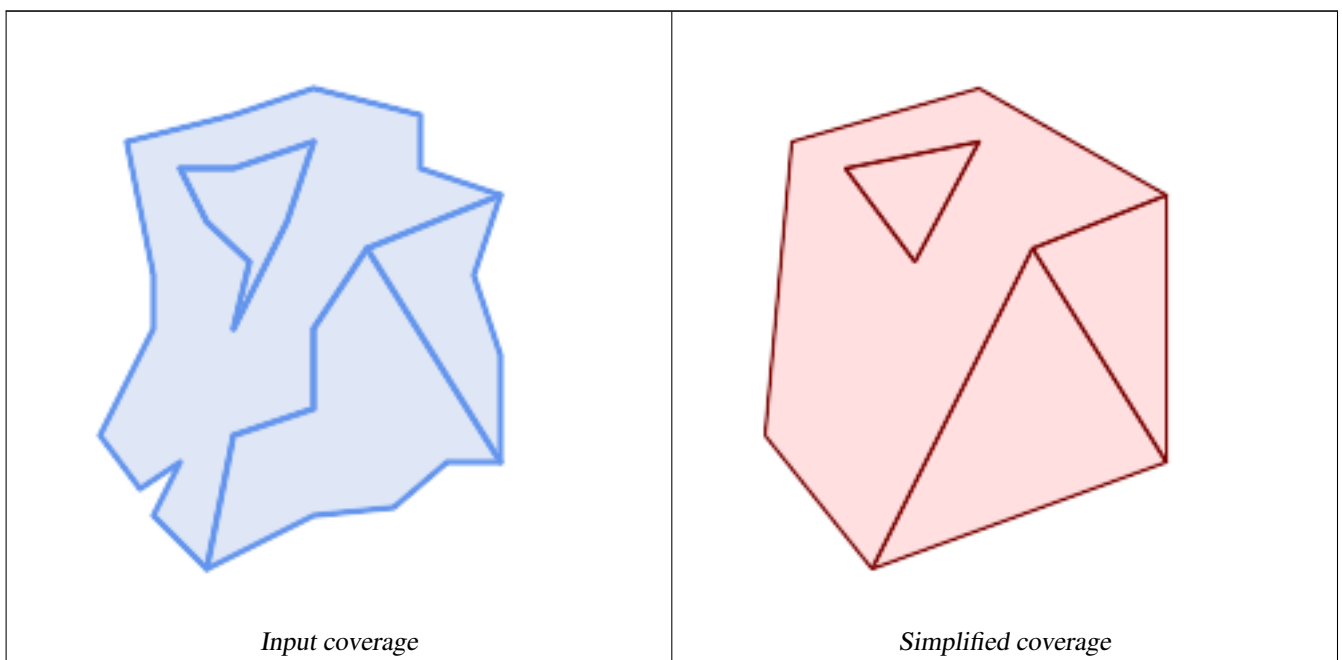
The simplification uses a variant of the [Visvalingam–Whyatt algorithm](#). The *tolerance* parameter has units of distance, and is roughly equal to the square root of triangular areas to be simplified.

To simplify only the "internal" edges of the coverage (those that are shared by two polygons) set the *simplifyBoundary* parameter to false.

**Note**

If the input is not a valid coverage there may be unexpected artifacts in the output (such as boundary intersections, or separated boundaries which appeared to be shared). Use [ST\\_CoverageInvalidEdges](#) to determine if a coverage is valid.

Availability: 3.4.0 - requires GEOS >= 3.12.0

**Examples**

```
WITH coverage(id, geom) AS (VALUES
 (1, 'POLYGON ((160 150, 110 130, 90 100, 90 70, 60 60, 50 10, 30 30, 40 50, 25 40, 10 60, ↵
 30 100, 30 120, 20 170, 60 180, 90 190, 130 180, 130 160, 160 150), (40 160, 50 140, ↵
 66 125, 60 100, 80 140, 90 170, 60 160, 40 160))'::geometry),
 (2, 'POLYGON ((40 160, 60 160, 90 170, 80 140, 60 100, 66 125, 50 140, 40 160))':: ↵
 geometry),
 (3, 'POLYGON ((110 130, 160 50, 140 50, 120 33, 90 30, 50 10, 60 60, 90 70, 90 100, 110 ↵
 130))'::geometry),
 (4, 'POLYGON ((160 150, 150 120, 160 90, 160 50, 110 130, 160 150))'::geometry)
)
SELECT id, ST_AsText(ST_CoverageSimplify(geom, 30) OVER ())
FROM coverage;
```

| id | st_astext                                                                                                  |
|----|------------------------------------------------------------------------------------------------------------|
| 1  | POLYGON ((160 150, 110 130, 50 10, 10 60, 20 170, 90 190, 160 150), (40 160, 66 125, ↵<br>90 170, 40 160)) |
| 2  | POLYGON ((40 160, 66 125, 90 170, 40 160))                                                                 |
| 3  | POLYGON ((110 130, 160 50, 50 10, 110 130))                                                                |
| 3  | POLYGON ((160 150, 160 50, 110 130, 160 150))                                                              |

See Also

[ST\\_CoverageInvalidEdges](#)

7.15.3 ST\_CoverageUnion

ST\_CoverageUnion — Computes the union of a set of polygons forming a coverage by removing shared edges.

Synopsis

geometry **ST\_CoverageUnion**(geometry set geom);

Description

An aggregate function which unions a set of polygons forming a polygonal coverage. The result is a polygonal geometry covering the same area as the coverage. This function produces the same result as [ST\\_Union](#), but uses the coverage structure to compute the union much faster.

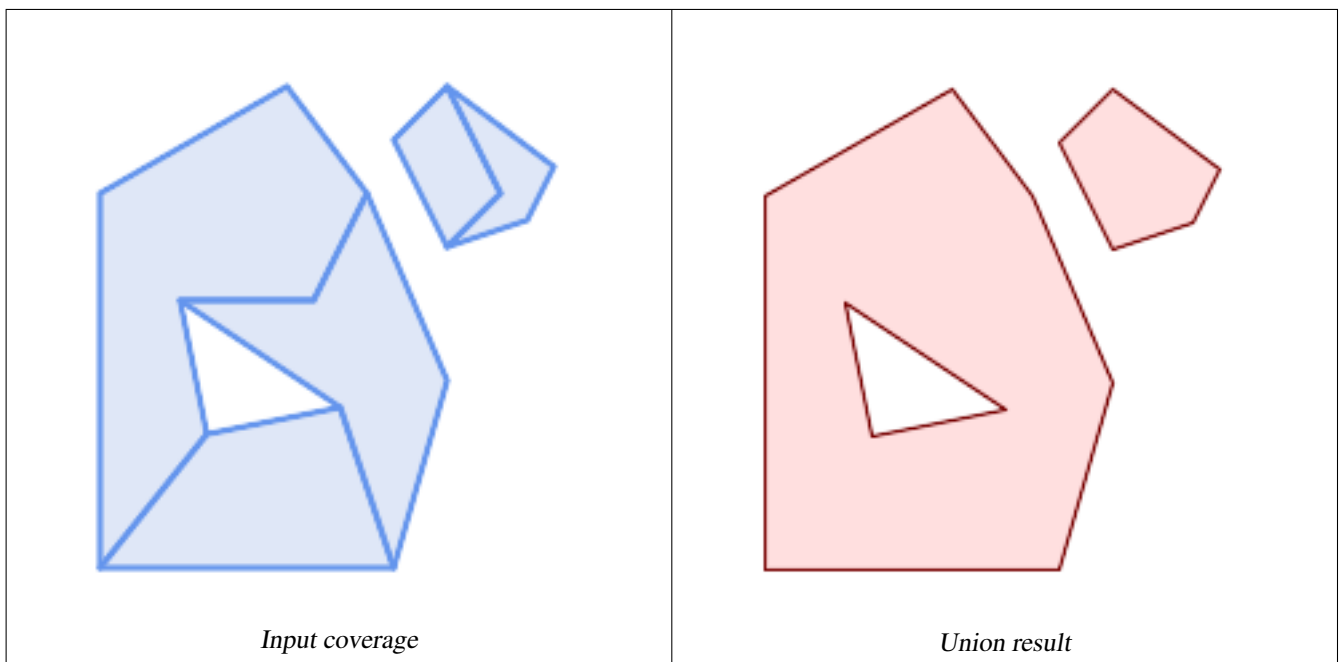


**Note**  
If the input is not a valid coverage there may be unexpected artifacts in the output (such as unmerged or overlapping polygons). Use [ST\\_CoverageInvalidEdges](#) to determine if a coverage is valid.

Availability: 3.4.0 - requires GEOS >= 3.8.0

Examples





```
WITH coverage(id, geom) AS (VALUES
 (1, 'POLYGON ((10 10, 10 150, 80 190, 110 150, 90 110, 40 110, 50 60, 10 10))'::geometry) ←
 /
 (2, 'POLYGON ((120 10, 10 10, 50 60, 100 70, 120 10))'::geometry),
 (3, 'POLYGON ((140 80, 120 10, 100 70, 40 110, 90 110, 110 150, 140 80))'::geometry),
 (4, 'POLYGON ((140 190, 120 170, 140 130, 160 150, 140 190))'::geometry),
 (5, 'POLYGON ((180 160, 170 140, 140 130, 160 150, 140 190, 180 160))'::geometry)
)
SELECT ST_AsText(ST_CoverageUnion(geom))
FROM coverage;

MULTIPOLYGON (((10 150, 80 190, 110 150, 140 80, 120 10, 10 10, 10 150), (50 60, 100 70, 40 ←
 110, 50 60)), ((120 170, 140 190, 180 160, 170 140, 140 130, 120 170)))
```

### See Also

[ST\\_CoverageInvalidEdges](#), [ST\\_Union](#)

## 7.16 Affine Transformations

### 7.16.1 ST\_Affine

**ST\_Affine** — Apply a 3D affine transformation to a geometry.

#### Synopsis

geometry **ST\_Affine**(geometry geomA, float a, float b, float c, float d, float e, float f, float g, float h, float i, float xoff, float yoff, float zoff);

geometry **ST\_Affine**(geometry geomA, float a, float b, float d, float e, float xoff, float yoff);

**Description**

Applies a 3D affine transformation to the geometry to do things like translate, rotate, scale in one step.

Version 1: The call

```
ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)
```

represents the transformation matrix

```
/ a b c xoff \
| d e f yoff |
| g h i zoff |
\ 0 0 0 1 /
```

and the vertices are transformed as follows:

```
x' = a*x + b*y + c*z + xoff
y' = d*x + e*y + f*z + yoff
z' = g*x + h*y + i*z + zoff
```

All of the translate / scale functions below are expressed via such an affine transformation.

Version 2: Applies a 2d affine transformation to the geometry. The call

```
ST_Affine(geom, a, b, d, e, xoff, yoff)
```

represents the transformation matrix

```
/ a b 0 xoff \ / a b xoff \
| d e 0 yoff | rsp. | d e yoff |
| 0 0 1 0 | \ 0 0 1 /
\ 0 0 0 1 /
```

and the vertices are transformed as follows:

```
x' = a*x + b*y + xoff
y' = d*x + e*y + yoff
z' = z
```

This method is a subcase of the 3D method above.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Availability: 1.1.2. Name changed from Affine to ST\_Affine in 1.2.2

**Note**

Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



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



This method supports Circular Strings and Curves.

## Examples

```
--Rotate a 3d line 180 degrees about the z axis. Note this is long-hand for doing ↵
ST_Rotate();
SELECT ST_AseWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), 0, 0, ↵
0, 1, 0, 0, 0)) As using_affine,
ST_AseWKT(ST_Rotate(geom, pi())) As using_rotate
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
using_affine | using_rotate
-----+-----
LINESTRING(-1 -2 3,-1 -4 3) | LINESTRING(-1 -2 3,-1 -4 3)
(1 row)

--Rotate a 3d line 180 degrees in both the x and z axis
SELECT ST_AseWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin(pi()) ↵
, 0, sin(pi()), cos(pi()), 0, 0, 0))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
st_asewkt

LINESTRING(-1 -2 -3,-1 -4 -3)
(1 row)
```

## See Also

[ST\\_Rotate](#), [ST\\_Scale](#), [ST\\_Translate](#), [ST\\_TransScale](#)

## 7.16.2 ST\_Rotate

**ST\_Rotate** — Rotates a geometry about an origin point.

### Synopsis

```
geometry ST_Rotate(geometry geomA, float rotRadians);
geometry ST_Rotate(geometry geomA, float rotRadians, float x0, float y0);
geometry ST_Rotate(geometry geomA, float rotRadians, geometry pointOrigin);
```

### Description

Rotates geometry *rotRadians* counter-clockwise about the origin point. The rotation origin can be specified either as a POINT geometry, or as x and y coordinates. If the origin is not specified, the geometry is rotated about POINT(0 0).

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Enhanced: 2.0.0 additional parameters for specifying the origin of rotation were added.

Availability: 1.1.2. Name changed from Rotate to ST\_Rotate in 1.2.2



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



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Examples

```
--Rotate 180 degrees
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()));
 st_asewkt

LINESTRING(-50 -160,-50 -50,-100 -50)
(1 row)

--Rotate 30 degrees counter-clockwise at x=50, y=160
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()/6, 50, 160));
 st_asewkt

LINESTRING(50 160,105 64.7372055837117,148.301270189222 89.7372055837117)
(1 row)

--Rotate 60 degrees clockwise from centroid
SELECT ST_AsEWKT(ST_Rotate(geom, -pi()/3, ST_Centroid(geom)))
FROM (SELECT 'LINESTRING (50 160, 50 50, 100 50)::geometry AS geom) AS foo;
 st_asewkt

LINESTRING(116.4225 130.6721,21.1597 75.6721,46.1597 32.3708)
(1 row)
```

## See Also

[ST\\_Affine](#), [ST\\_RotateX](#), [ST\\_RotateY](#), [ST\\_RotateZ](#)

### 7.16.3 ST\_RotateX

ST\_RotateX — Rotates a geometry about the X axis.

#### Synopsis

geometry **ST\_RotateX**(geometry geomA, float rotRadians);

#### Description

Rotates a geometry geomA - rotRadians about the X axis.



#### Note

ST\_RotateX(geomA, rotRadians) is short-hand for ST\_Affine(geomA, 1, 0, 0, 0, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0).

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Availability: 1.1.2. Name changed from RotateX to ST\_RotateX in 1.2.2



This function supports Polyhedral surfaces.



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



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Examples

```
--Rotate a line 90 degrees along x-axis
SELECT ST_AsEWKT(ST_RotateX(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
 st_asewkt

LINESTRING(1 -3 2,1 -1 1)
```

## See Also

[ST\\_Affine](#), [ST\\_RotateY](#), [ST\\_RotateZ](#)

### 7.16.4 ST\_RotateY

ST\_RotateY — Rotates a geometry about the Y axis.

## Synopsis

geometry **ST\_RotateY**(geometry geomA, float rotRadians);

## Description

Rotates a geometry geomA - rotRadians about the y axis.



### Note

ST\_RotateY(geomA, rotRadians) is short-hand for ST\_Affine(geomA, cos(rotRadians), 0, sin(rotRadians), 0, 1, 0, -sin(rotRadians), 0, cos(rotRadians), 0, 0, 0).

Availability: 1.1.2. Name changed from RotateY to ST\_RotateY in 1.2.2

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.



This function supports Polyhedral surfaces.



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



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Examples

```
--Rotate a line 90 degrees along y-axis
SELECT ST_AsEWKT(ST_RotateY(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
 st_asewkt

LINESTRING(3 2 -1,1 1 -1)
```

## See Also

[ST\\_Affine](#), [ST\\_RotateX](#), [ST\\_RotateZ](#)

### 7.16.5 ST\_RotateZ

ST\_RotateZ — Rotates a geometry about the Z axis.

#### Synopsis

geometry **ST\_RotateZ**(geometry geomA, float rotRadians);

#### Description

Rotates a geometry geomA - rotRadians about the Z axis.



#### Note

This is a synonym for ST\_Rotate



#### Note

ST\_RotateZ(geomA, rotRadians) is short-hand for SELECT ST\_Affine(geomA, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0, 1, 0, 0, 0).

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Availability: 1.1.2. Name changed from RotateZ to ST\_RotateZ in 1.2.2



#### Note

Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+



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



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

#### Examples

```
--Rotate a line 90 degrees along z-axis
SELECT ST_AsEWKT(ST_RotateZ(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
 st_asewkt

LINESTRING(-2 1 3,-1 1 1)

--Rotate a curved circle around z-axis
SELECT ST_AsEWKT(ST_RotateZ(geom, pi()/2))
FROM (SELECT ST_LineToCurve(ST_Buffer(ST_GeomFromText('POINT(234 567)'), 3)) As geom) As foo;
```

---

```
CURVEPOLYGON(CIRCULARSTRING(-567 237,-564.87867965644 236.12132034356,-564 234,-569.12132034356 231.87867965644,-567 237))
```

**See Also**

[ST\\_Affine](#), [ST\\_RotateX](#), [ST\\_RotateY](#)

**7.16.6 ST\_Scale**

**ST\_Scale** — Scales a geometry by given factors.

**Synopsis**

```
geometry ST_Scale(geometry geomA, float XFactor, float YFactor, float ZFactor);
geometry ST_Scale(geometry geomA, float XFactor, float YFactor);
geometry ST_Scale(geometry geom, geometry factor);
geometry ST_Scale(geometry geom, geometry factor, geometry origin);
```

**Description**

Scales the geometry to a new size by multiplying the ordinates with the corresponding factor parameters.

The version taking a geometry as the `factor` parameter allows passing a 2d, 3dm, 3dz or 4d point to set scaling factor for all supported dimensions. Missing dimensions in the `factor` point are equivalent to no scaling the corresponding dimension.

The three-geometry variant allows a "false origin" for the scaling to be passed in. This allows "scaling in place", for example using the centroid of the geometry as the false origin. Without a false origin, scaling takes place relative to the actual origin, so all coordinates are just multiplied by the scale factor.

**Note**

Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

Availability: 1.1.0.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Enhanced: 2.2.0 support for scaling all dimension (`factor` parameter) was introduced.

Enhanced: 2.5.0 support for scaling relative to a local origin (`origin` parameter) was introduced.



This function supports Polyhedral surfaces.



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



This method supports Circular Strings and Curves.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



This function supports M coordinates.

## Examples

```
--Version 1: scale X, Y, Z
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75, 0.8));
 st_asewkt

LINESTRING(0.5 1.5 2.4,0.5 0.75 0.8)

--Version 2: Scale X Y
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75));
 st_asewkt

LINESTRING(0.5 1.5 3,0.5 0.75 1)

--Version 3: Scale X Y Z M
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)'),
 ST_MakePoint(0.5, 0.75, 2, -1)));
 st_asewkt

LINESTRING(0.5 1.5 6 -4,0.5 0.75 2 -1)

--Version 4: Scale X Y using false origin
SELECT ST_AsText(ST_Scale('LINESTRING(1 1, 2 2)', 'POINT(2 2)', 'POINT(1 1)::geometry'));
 st_astext

LINESTRING(1 1,3 3)
```

## See Also

[ST\\_Affine](#), [ST\\_TransScale](#)

## 7.16.7 ST\_Translate

ST\_Translate — Translates a geometry by given offsets.

### Synopsis

```
geometry ST_Translate(geometry g1, float deltax, float deltay);
geometry ST_Translate(geometry g1, float deltax, float deltay, float deltaz);
```

### Description

Returns a new geometry whose coordinates are translated delta x,delta y,delta z units. Units are based on the units defined in spatial reference (SRID) for this geometry.



#### Note

Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

Availability: 1.2.2



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



This method supports Circular Strings and Curves.



## Examples

### Move a point 1 degree longitude

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('POINT(-71.01 42.37)',4326),1,0)) As ↵
 wgs_transgeomtxt;

 wgs_transgeomtxt

 POINT(-70.01 42.37)
```

### Move a linestring 1 degree longitude and 1/2 degree latitude

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('LINESTRING(-71.01 42.37,-71.11 42.38)',4326) ↵
 ,1,0.5)) As wgs_transgeomtxt;
 wgs_transgeomtxt

 LINESTRING(-70.01 42.87,-70.11 42.88)
```

### Move a 3d point

```
SELECT ST_AsEWKT(ST_Translate(CAST('POINT(0 0 0)' As geometry), 5, 12,3));
 st_asewkt

 POINT(5 12 3)
```

### Move a curve and a point

```
SELECT ST_AsText(ST_Translate(ST_Collect('CURVEPOLYGON(CIRCULARSTRING(4 3,3.12 0.878,1 ↵
 0,-1.121 5.1213,6 7, 8 9,4 3))','POINT(1 3)'),1,2));

GEOMETRYCOLLECTION(CURVEPOLYGON(CIRCULARSTRING(5 5,4.12 2.878,2 2,-0.121 7.1213,7 9,9 11,5 ↵
 5)),POINT(2 5))
```

## See Also

[ST\\_Affine](#), [ST\\_AsText](#), [ST\\_GeomFromText](#)

## 7.16.8 ST\_TransScale

**ST\_TransScale** — Translates and scales a geometry by given offsets and factors.

### Synopsis

geometry **ST\_TransScale**(geometry geomA, float deltaX, float deltaY, float XFactor, float YFactor);

### Description

Translates the geometry using the deltaX and deltaY args, then scales it using the XFactor, YFactor args, working in 2D only.



#### Note

**ST\_TransScale**(geomA, deltaX, deltaY, XFactor, YFactor) is short-hand for **ST\_Affine**(geomA, XFactor, 0, 0, 0, YFactor, 0, 0, 0, 1, deltaX\*XFactor, deltaY\*YFactor, 0).

**Note**

Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

Availability: 1.1.0.



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



This method supports Circular Strings and Curves.

**Examples**

```
SELECT ST_AsEWKT(ST_TransScale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 1, 1, 2));
 st_asewkt

LINESTRING(1.5 6 3,1.5 4 1)

--Buffer a point to get an approximation of a circle, convert to curve and then translate ↩
 1,2 and scale it 3,4
SELECT ST_AsText(ST_Transscale(ST_LineToCurve(ST_Buffer('POINT(234 567)', 3)),1,2,3,4));

CURVEPOLYGON(CIRCULARSTRING(714 2276,711.363961030679 2267.51471862576,705 ↩
 2264,698.636038969321 2284.48528137424,714 2276))
```

**See Also**

[ST\\_Affine](#), [ST\\_Translate](#)

## 7.17 Clustering Functions

### 7.17.1 ST\_ClusterDBSCAN

**ST\_ClusterDBSCAN** — Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.

**Synopsis**

integer **ST\_ClusterDBSCAN**(geometry winset geom, float8 eps, integer minpoints);

**Description**

Returns cluster number for each input geometry, based on a 2D implementation of the [Density-based spatial clustering of applications with noise \(DBSCAN\)](#) algorithm. Unlike [ST\\_ClusterKMeans](#), it does not require the number of clusters to be specified, but instead uses the desired [distance](#) (eps) and density (minpoints) parameters to construct each cluster.

An input geometry will be added to a cluster if it is either:

- A "core" geometry, that is within `eps distance` of at least `minpoints` input geometries (including itself) or
- A "border" geometry, that is within `eps distance` of a core geometry.

Note that border geometries may be within `eps` distance of core geometries in more than one cluster; in this case, either assignment would be correct, and the border geometry will be arbitrarily assigned to one of the available clusters. In these cases, it is possible for a correct cluster to be generated with fewer than `minpoints` geometries. When assignment of a border geometry is ambiguous, repeated calls to `ST_ClusterDBSCAN` will produce identical results if an `ORDER BY` clause is included in the window definition, but cluster assignments may differ from other implementations of the same algorithm.

**Note**

Input geometries that do not meet the criteria to join any other cluster will be assigned a cluster number of `NULL`.

---

Availability: 2.3.0

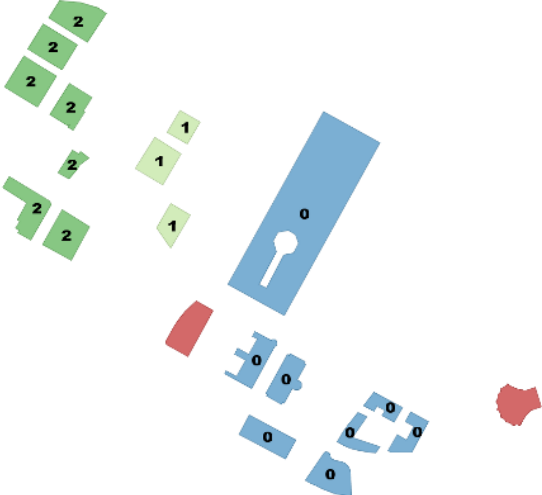


This method supports Circular Strings and Curves.

**Examples**

Assigning a cluster number to each polygon within 50 meters of each other. Require at least 2 polygons per cluster

---



*within 50 meters at least 2 per cluster. singletons have NULL for cid*

```
SELECT name, ST_ClusterDBSCAN(geom, eps ↵ := 50, minpoints := 2) over () AS cid FROM boston_polys WHERE name > '' AND building > '' AND ST_DWithin(geom, ST_Transform(ST_GeomFromText('POINT ↵ (-71.04054 42.35141)', 4326), 26986), 500);
```

| name                                | ↵ |
|-------------------------------------|---|
| bucket                              |   |
| Manulife Tower                      | ↵ |
| 0                                   |   |
| Park Lane Seaport I                 | ↵ |
| 0                                   |   |
| Park Lane Seaport II                | ↵ |
| 0                                   |   |
| Renaissance Boston Waterfront Hotel | ↵ |
| 0                                   |   |
| Seaport Boston Hotel                | ↵ |
| 0                                   |   |
| Seaport Hotel & World Trade Center  | ↵ |
| 0                                   |   |
| Waterside Place                     | ↵ |
| 0                                   |   |
| World Trade Center East             | ↵ |
| 0                                   |   |
| 100 Northern Avenue                 | ↵ |
| 1                                   |   |
| 100 Pier 4                          | ↵ |
| 1                                   |   |
| The Institute of Contemporary Art   | ↵ |
| 1                                   |   |
| 101 Seaport                         | ↵ |
| 2                                   |   |
| District Hall                       | ↵ |
| 2                                   |   |
| One Marina Park Drive               | ↵ |
| 2                                   |   |
| Twenty Two Liberty                  | ↵ |
| 2                                   |   |
| Vertex                              | ↵ |
| 2                                   |   |
| Vertex                              | ↵ |
| 2                                   |   |
| Watermark Seaport                   | ↵ |
| 2                                   |   |
| Blue Hills Bank Pavilion            | ↵ |
| NULL                                |   |
| World Trade Center West             | ↵ |
| NULL                                |   |
| (20 rows)                           |   |

Combining parcels with the same cluster number into a single geometry. This uses named argument calling

```
SELECT cid, ST_Collect(geom) AS cluster_geom, array_agg(parcel_id) AS ids_in_cluster FROM (
SELECT parcel_id, ST_ClusterDBSCAN(geom, eps := 0.5, minpoints := 5) over () AS cid, ↵
geom
FROM parcels) sq
GROUP BY cid;
```

See Also

[ST\\_DWithin](#), [ST\\_ClusterKMeans](#), [ST\\_ClusterIntersecting](#), [ST\\_ClusterWithin](#) [ST\\_ClusterIntersectingWin](#)

### 7.17.2 ST\_ClusterIntersectingWin

**ST\_ClusterIntersectingWin** — Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.

#### Synopsis

integer **ST\_ClusterIntersectingWin**(geometry winset geom);

#### Description

**ST\_ClusterIntersectingWin** is a windowing function that builds inter-connecting clusters of geometries that intersect. It is possible to traverse all geometries in a cluster without leaving the cluster. The return value is the cluster number that the geometry argument participates in, or null for null inputs.

Availability: 3.4.0

#### Examples

```
WITH testdata AS (
 SELECT id, geom::geometry FROM (
 VALUES (1, 'LINESTRING (0 0, 1 1)'),
 (2, 'LINESTRING (5 5, 4 4)'),
 (3, 'LINESTRING (6 6, 7 7)'),
 (4, 'LINESTRING (0 0, -1 -1)'),
 (5, 'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))')) AS t(id, geom)
)
SELECT id,
 ST_AsText(geom),
 ST_ClusterIntersectingWin(geom) OVER () AS cluster
FROM testdata;
```

| id | st_astext                      | cluster |
|----|--------------------------------|---------|
| 1  | LINESTRING(0 0,1 1)            | 0       |
| 2  | LINESTRING(5 5,4 4)            | 0       |
| 3  | LINESTRING(6 6,7 7)            | 1       |
| 4  | LINESTRING(0 0,-1 -1)          | 0       |
| 5  | POLYGON((0 0,4 0,4 4,0 4,0 0)) | 0       |

#### See Also

[ST\\_ClusterDBSCAN](#), [ST\\_ClusterKMeans](#), [ST\\_ClusterWithinWin](#) [ST\\_ClusterWithin](#) [ST\\_ClusterIntersecting](#)

### 7.17.3 ST\_ClusterIntersecting

**ST\_ClusterIntersecting** — Aggregate function that clusters input geometries into connected sets.

#### Synopsis

geometry[] **ST\_ClusterIntersecting**(geometry set g);

## Description

`ST_ClusterIntersecting` is an aggregate function that returns an array of `GeometryCollections`, where each `GeometryCollection` represents an interconnected set of geometries.

Availability: 2.2.0

## Examples

```
WITH testdata AS
 (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)::geometry',
 'LINESTRING (5 5, 4 4)::geometry',
 'LINESTRING (6 6, 7 7)::geometry',
 'LINESTRING (0 0, -1 -1)::geometry',
 'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))::geometry']) AS geom)

SELECT ST_AsText(unnest(ST_ClusterIntersecting(geom))) FROM testdata;

--result

st_astext

GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

## See Also

[ST\\_ClusterDBSCAN](#), [ST\\_ClusterKMeans](#), [ST\\_ClusterWithin](#), [ST\\_ClusterWithin](#)

### 7.17.4 ST\_ClusterKMeans

`ST_ClusterKMeans` — Window function that returns a cluster id for each input geometry using the K-means algorithm.

## Synopsis

integer **ST\_ClusterKMeans**(geometry winset geom, integer number\_of\_clusters, float max\_radius);

## Description

Returns **K-means** cluster number for each input geometry. The distance used for clustering is the distance between the centroids for 2D geometries, and distance between bounding box centers for 3D geometries. For POINT inputs, M coordinate will be treated as weight of input and has to be larger than 0.

`max_radius`, if set, will cause `ST_ClusterKMeans` to generate more clusters than `k` ensuring that no cluster in output has radius larger than `max_radius`. This is useful in reachability analysis.

Enhanced: 3.2.0 Support for `max_radius`

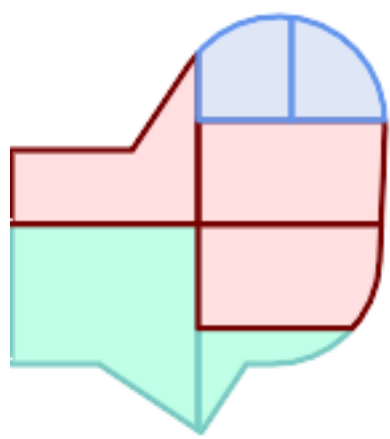
Enhanced: 3.1.0 Support for 3D geometries and weights

Availability: 2.3.0

Examples

Generate dummy set of parcels for examples:

```
CREATE TABLE parcels AS
SELECT lpad((row_number() over())::text,3,'0') As parcel_id, geom,
('{residential, commercial}'::text[])[1 + mod(row_number()OVER(),2)] As type
FROM
 ST_Subdivide(ST_Buffer('SRID=3857;LINESTRING(40 100, 98 100, 100 150, 60 90)'::geometry ←
 40, 'endcap=square'),12) As geom;
```



Parcels color-coded by cluster number (cid)

```
SELECT ST_ClusterKMeans(geom, 3) OVER() AS cid, parcel_id, geom
FROM parcels;
```

| cid | parcel_id | geom          |
|-----|-----------|---------------|
| 0   | 001       | 0103000000... |
| 0   | 002       | 0103000000... |
| 1   | 003       | 0103000000... |
| 0   | 004       | 0103000000... |
| 1   | 005       | 0103000000... |
| 2   | 006       | 0103000000... |
| 2   | 007       | 0103000000... |

Partitioning parcel clusters by type:

```
SELECT ST_ClusterKMeans(geom, 3) over (PARTITION BY type) AS cid, parcel_id, type
FROM parcels;
```

| cid | parcel_id | type        |
|-----|-----------|-------------|
| 1   | 005       | commercial  |
| 1   | 003       | commercial  |
| 2   | 007       | commercial  |
| 0   | 001       | commercial  |
| 1   | 004       | residential |
| 0   | 002       | residential |
| 2   | 006       | residential |

Example: Clustering a preaggregated planetary-scale data population dataset using 3D clustering and weighting. Identify at least 20 regions based on **Kontur Population Data** that do not span more than 3000 km from their center:

```
create table kontur_population_3000km_clusters as
select
 geom,
 ST_ClusterKMeans(
 ST_Force4D(
 ST_Transform(ST_Force3D(geom), 4978), -- cluster in 3D XYZ CRS
 mvalue := population -- set clustering to be weighed by population
),
 20, -- aim to generate at least 20 clusters
 max_radius := 3000000 -- but generate more to make each under 3000 km radius
) over () as cid
from
 kontur_population;
```



*World population clustered to above specs produces 46 clusters. Clusters are centered at well-populated regions (New York, Moscow). Greenland is one cluster. There are island clusters that span across the antimeridian. Cluster edges follow Earth's curvature.*

#### See Also

[ST\\_ClusterDBSCAN](#), [ST\\_ClusterIntersectingWin](#), [ST\\_ClusterWithinWin](#), [ST\\_ClusterIntersecting](#), [ST\\_ClusterWithin](#), [ST\\_Subdivide](#), [ST\\_Force3D](#), [ST\\_Force4D](#),

### 7.17.5 ST\_ClusterWithin

**ST\_ClusterWithin** — Aggregate function that clusters input geometries by separation distance.

#### Synopsis

```
geometry[] ST_ClusterWithin(geometry set g, float8 distance);
```

#### Description

**ST\_ClusterWithin** is an aggregate function that returns an array of **GeometryCollections**, where each **GeometryCollection** represents a set of geometries separated by no more than the specified distance. (Distances are Cartesian distances in the units of the SRID.)

Availability: 2.2.0



This method supports Circular Strings and Curves.



## Examples

```
WITH testdata AS
 (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)::geometry',
 'LINESTRING (5 5, 4 4)::geometry',
 'LINESTRING (6 6, 7 7)::geometry',
 'LINESTRING (0 0, -1 -1)::geometry',
 'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))::geometry']) AS geom)

SELECT ST_AsText(unnest(ST_ClusterWithin(geom, 1.4))) FROM testdata;

--result

st_astext

GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

## See Also

[ST\\_ClusterDBSCAN](#), [ST\\_ClusterKMeans](#), [ST\\_ClusterIntersectingWin](#), [ST\\_ClusterWithinWin](#), [ST\\_ClusterIntersecting](#)

## 7.17.6 ST\_ClusterWithinWin

**ST\_ClusterWithinWin** — Window function that returns a cluster id for each input geometry, clustering using separation distance.

### Synopsis

integer **ST\_ClusterWithinWin**(geometry winset geom, float8 distance);

### Description

**ST\_ClusterWithinWin** is a window function that returns an integer for each input geometry, where the integer the cluster number the geometry is a member of. Clusters represent a set of input geometries separated by no more than the specified distance. (Distances are Cartesian distances in the units of the SRID.)

**ST\_ClusterWithinWin** is equivalent to running [ST\\_ClusterDBSCAN](#) with a `minpoints` of zero.

Availability: 3.4.0



This method supports Circular Strings and Curves.

## Examples

```
WITH testdata AS (
 SELECT id, geom::geometry FROM (
 VALUES (1, 'LINESTRING (0 0, 1 1)'),
 (2, 'LINESTRING (5 5, 4 4)'),
 (3, 'LINESTRING (6 6, 7 7)'),
 (4, 'LINESTRING (0 0, -1 -1)'),
 (5, 'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))') AS t(id, geom)
)
)
SELECT id,
 ST_AsText(geom),
 ST_ClusterWithinWin(geom, 1.4) OVER () AS cluster
```

```
FROM testdata;
```

| id | st_astext                      | cluster |
|----|--------------------------------|---------|
| 1  | LINESTRING(0 0,1 1)            | 0       |
| 2  | LINESTRING(5 5,4 4)            | 0       |
| 3  | LINESTRING(6 6,7 7)            | 1       |
| 4  | LINESTRING(0 0,-1 -1)          | 0       |
| 5  | POLYGON((0 0,4 0,4 4,0 4,0 0)) | 0       |

**See Also**

[ST\\_ClusterDBSCAN](#), [ST\\_ClusterKMeans](#), [ST\\_ClusterIntersectingWin](#), [ST\\_ClusterWithin](#), [ST\\_ClusterIntersecting](#)

## 7.18 Bounding Box Functions

### 7.18.1 Box2D

Box2D — Returns a BOX2D representing the 2D extent of a geometry.

**Synopsis**

box2d **Box2D**(geometry geom);

**Description**

Returns a **box2d** representing the 2D extent of the geometry.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

-  This method supports Circular Strings and Curves.
-  This function supports Polyhedral surfaces.
-  This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**Examples**

```
SELECT Box2D(ST_GeomFromText('LINESTRING(1 2, 3 4, 5 6)'));
```

```
box2d

BOX(1 2,5 6)
```

```
SELECT Box2D(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'));
```

```
box2d

BOX(220186.984375 150406,220288.25 150506.140625)
```

**See Also**

[Box3D](#), [ST\\_GeomFromText](#)

**7.18.2 Box3D**

Box3D — Returns a BOX3D representing the 3D extent of a geometry.

**Synopsis**

```
box3d Box3D(geometry geom);
```

**Description**

Returns a [box3d](#) representing the 3D extent of the geometry.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



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

**Examples**

```
SELECT Box3D(ST_GeomFromEWKT('LINESTRING(1 2 3, 3 4 5, 5 6 5)'));
```

```
Box3d
```

```

```

```
BOX3D(1 2 3,5 6 5)
```

```
SELECT Box3D(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 1,220227 150406 1)'));
```

```
Box3d
```

```

```

```
BOX3D(220227 150406 1,220268 150415 1)
```

**See Also**

[Box2D](#), [ST\\_GeomFromEWKT](#)

**7.18.3 ST\_EstimatedExtent**

ST\_EstimatedExtent — Returns the estimated extent of a spatial table.

**Synopsis**

```
box2d ST_EstimatedExtent(text schema_name, text table_name, text geocolumn_name, boolean parent_only);
```

```
box2d ST_EstimatedExtent(text schema_name, text table_name, text geocolumn_name);
```

```
box2d ST_EstimatedExtent(text table_name, text geocolumn_name);
```

## Description

Returns the estimated extent of a spatial table as a **box2d**. The current schema is used if not specified. The estimated extent is taken from the geometry column's statistics. This is usually much faster than computing the exact extent of the table using **ST\_Extent** or **ST\_3DExtent**.

The default behavior is to also use statistics collected from child tables (tables with INHERITS) if available. If `parent_only` is set to TRUE, only statistics for the given table are used and child tables are ignored.

For PostgreSQL >= 8.0.0 statistics are gathered by VACUUM ANALYZE and the result extent will be about 95% of the actual one. For PostgreSQL < 8.0.0 statistics are gathered by running `update_geometry_stats()` and the result extent is exact.



### Note

In the absence of statistics (empty table or no ANALYZE called) this function returns NULL. Prior to version 1.5.4 an exception was thrown instead.

Availability: 1.0.0

Changed: 2.1.0. Up to 2.0.x this was called `ST_Estimated_Extent`.



This method supports Circular Strings and Curves.

## Examples

```
SELECT ST_EstimatedExtent('ny', 'edges', 'geom');
--result--
BOX(-8877653 4912316,-8010225.5 5589284)

SELECT ST_EstimatedExtent('feature_poly', 'geom');
--result--
BOX(-124.659652709961 24.6830825805664,-67.7798080444336 49.0012092590332)
```

## See Also

**ST\_Extent**, **ST\_3DExtent**

## 7.18.4 ST\_Expand

**ST\_Expand** — Returns a bounding box expanded from another bounding box or a geometry.

### Synopsis

```
geometry ST_Expand(geometry geom, float units_to_expand);
geometry ST_Expand(geometry geom, float dx, float dy, float dz=0, float dm=0);
box2d ST_Expand(box2d box, float units_to_expand);
box2d ST_Expand(box2d box, float dx, float dy);
box3d ST_Expand(box3d box, float units_to_expand);
box3d ST_Expand(box3d box, float dx, float dy, float dz=0);
```

## Description

Returns a bounding box expanded from the bounding box of the input, either by specifying a single distance with which the box should be expanded on both axes, or by specifying an expansion distance for each axis. Uses double-precision. Can be used for distance queries, or to add a bounding box filter to a query to take advantage of a spatial index.

In addition to the version of `ST_Expand` accepting and returning a geometry, variants are provided that accept and return **box2d** and **box3d** data types.

Distances are in the units of the spatial reference system of the input.

`ST_Expand` is similar to **ST\_Buffer**, except while buffering expands a geometry in all directions, `ST_Expand` expands the bounding box along each axis.



### Note

Pre version 1.3, `ST_Expand` was used in conjunction with **ST\_Distance** to do indexable distance queries. For example, `geom && ST_Expand('POINT(10 20)', 10) AND ST_Distance(geom, 'POINT(10 20)') < 10`. This has been replaced by the simpler and more efficient **ST\_DWithin** function.

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

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Examples



### Note

Examples below use US National Atlas Equal Area (SRID=2163) which is a meter projection

```
--10 meter expanded box around bbox of a linestring
SELECT CAST(ST_Expand(ST_GeomFromText('LINESTRING(2312980 110676,2312923 110701,2312892 110714)', 2163),10) As box2d);
 st_expand

BOX(2312882 110666,2312990 110724)

--10 meter expanded 3D box of a 3D box
SELECT ST_Expand(CAST('BOX3D(778783 2951741 1,794875 2970042.61545891 10)' As box3d),10)
 st_expand

BOX3D(778773 2951731 -9,794885 2970052.61545891 20)

--10 meter geometry astext rep of a expand box around a point geometry
SELECT ST_AsEWKT(ST_Expand(ST_GeomFromEWKT('SRID=2163;POINT(2312980 110676)'),10));
 st_asewkt

SRID=2163;POLYGON((2312970 110666,2312970 110686,2312990 110686,2312990 110666,2312970 110666))
```

**See Also**

[ST\\_Buffer](#), [ST\\_DWithin](#), [ST\\_SRID](#)

**7.18.5 ST\_Extent**

**ST\_Extent** — Aggregate function that returns the bounding box of geometries.

**Synopsis**

box2d **ST\_Extent**(geometry set geomfield);

**Description**

An aggregate function that returns a **box2d** bounding box that bounds a set of geometries.

The bounding box coordinates are in the spatial reference system of the input geometries.

**ST\_Extent** is similar in concept to Oracle Spatial/Locator's SDO\_AGGR\_MBR.

**Note**

**ST\_Extent** returns boxes with only X and Y ordinates even with 3D geometries. To return XYZ ordinates use [ST\\_3DExtent](#).

**Note**

The returned **box3d** value does not include a SRID. Use [ST\\_SetSRID](#) to convert it into a geometry with SRID meta-data. The SRID is the same as the input geometries.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**Examples****Note**

Examples below use Massachusetts State Plane ft (SRID=2249)

```
SELECT ST_Extent(geom) as bextent FROM sometable;
 st_bextent

BOX(739651.875 2908247.25,794875.8125 2970042.75)

--Return extent of each category of geometries
SELECT ST_Extent(geom) as bextent
FROM sometable
```

```
GROUP BY category ORDER BY category;

 bextent | name
-----+-----
BOX(778783.5625 2951741.25,794875.8125 2970042.75) | A
BOX(751315.8125 2919164.75,765202.6875 2935417.25) | B
BOX(739651.875 2917394.75,756688.375 2935866) | C

--Force back into a geometry
-- and render the extended text representation of that geometry
SELECT ST_SetSRID(ST_Extent(geom),2249) as bextent FROM sometable;

 bextent

SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,
794875.8125 2908247.25,739651.875 2908247.25))
```

See Also

[ST\\_EstimatedExtent](#), [ST\\_3DExtent](#), [ST\\_SetSRID](#)

7.18.6 ST\_3DExtent

ST\_3DExtent — Aggregate function that returns the 3D bounding box of geometries.

Synopsis

box3d **ST\_3DExtent**(geometry set geomfield);

Description

An aggregate function that returns a **box3d** (includes Z ordinate) bounding box that bounds a set of geometries. The bounding box coordinates are in the spatial reference system of the input geometries.



**Note**  
The returned `box3d` value does not include a SRID. Use [ST\\_SetSRID](#) to convert it into a geometry with SRID meta-data. The SRID is the same as the input geometries.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Changed: 2.0.0 In prior versions this used to be called ST\_Extent3D

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Examples

```
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_MakePoint(x,y,z) As geom
 FROM generate_series(1,3) As x
 CROSS JOIN generate_series(1,2) As y
 CROSS JOIN generate_series(0,2) As z) As foo;

b3extent

BOX3D(1 1 0,3 2 2)

--Get the extent of various elevated circular strings
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_Translate(ST_Force_3DZ(ST_LineToCurve(ST_Buffer(ST_Point(x,y),1))),0,0,z) ←
 As geom
 FROM generate_series(1,3) As x
 CROSS JOIN generate_series(1,2) As y
 CROSS JOIN generate_series(0,2) As z) As foo;

b3extent

BOX3D(1 0 0,4 2 2)
```

## See Also

[ST\\_Extent](#), [ST\\_Force3DZ](#), [ST\\_SetSRID](#)

## 7.18.7 ST\_MakeBox2D

**ST\_MakeBox2D** — Creates a BOX2D defined by two 2D point geometries.

### Synopsis

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

### Description

Creates a **box2d** defined by two Point geometries. This is useful for doing range queries.

## Examples

```
--Return all features that fall reside or partly reside in a US national atlas coordinate ←
 bounding box
--It is assumed here that the geometries are stored with SRID = 2163 (US National atlas ←
 equal area)
SELECT feature_id, feature_name, geom
FROM features
WHERE geom && ST_SetSRID(ST_MakeBox2D(ST_Point(-989502.1875, 528439.5625),
 ST_Point(-987121.375 ,529933.1875)),2163)
```

## See Also

[ST\\_Point](#), [ST\\_SetSRID](#), [ST\\_SRID](#)



### 7.18.8 ST\_3DMakeBox

ST\_3DMakeBox — Creates a BOX3D defined by two 3D point geometries.

#### Synopsis

```
box3d ST_3DMakeBox(geometry point3DLowLeftBottom, geometry point3DUpRightTop);
```

#### Description

Creates a **box3d** defined by two 3D Point geometries.



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

Changed: 2.0.0 In prior versions this used to be called ST\_MakeBox3D

#### Examples

```
SELECT ST_3DMakeBox(ST_MakePoint(-989502.1875, 528439.5625, 10),
 ST_MakePoint(-987121.375, 529933.1875, 10)) As abb3d

--bb3d--

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

#### See Also

[ST\\_MakePoint](#), [ST\\_SetSRID](#), [ST\\_SRID](#)

### 7.18.9 ST\_XMax

ST\_XMax — Returns the X maxima of a 2D or 3D bounding box or a geometry.

#### Synopsis

```
float ST_XMax(box3d aGeomorBox2DorBox3D);
```

#### Description

Returns the X maxima of a 2D or 3D bounding box or a geometry.



#### Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However, it will not accept a geometry or box2d text representation, since those do not auto-cast.



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



This method supports Circular Strings and Curves.

## Examples

```
SELECT ST_XMax('BOX3D(1 2 3, 4 5 6)');
st_xmax

4

SELECT ST_XMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmax

5

SELECT ST_XMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmax

3
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
a BOX3D
SELECT ST_XMax('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
150406 3)'));
st_xmax

220288.248780547
```

## See Also

[ST\\_XMin](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 7.18.10 ST\_XMin

**ST\_XMin** — Returns the X minima of a 2D or 3D bounding box or a geometry.

#### Synopsis

float **ST\_XMin**(box3d aGeomorBox2DorBox3D);

#### Description

Returns the X minima of a 2D or 3D bounding box or a geometry.



#### Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



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



This method supports Circular Strings and Curves.

## Examples

```
SELECT ST_XMin('BOX3D(1 2 3, 4 5 6)');
st_xmin

1

SELECT ST_XMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmin

1

SELECT ST_XMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmin

-3
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
 a BOX3D
SELECT ST_XMin('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
 150406 3)'));
st_xmin

220186.995121892
```

## See Also

[ST\\_XMax](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 7.18.11 ST\_YMax

**ST\_YMax** — Returns the Y maxima of a 2D or 3D bounding box or a geometry.

#### Synopsis

float **ST\_YMax**(box3d aGeomorBox2DorBox3D);

#### Description

Returns the Y maxima of a 2D or 3D bounding box or a geometry.



#### Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



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



This method supports Circular Strings and Curves.

## Examples

```
SELECT ST_YMax('BOX3D(1 2 3, 4 5 6)');
st_ymax

5

SELECT ST_YMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_ymax

6

SELECT ST_YMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_ymax

4
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
a BOX3D
SELECT ST_YMax('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
150406 3)'));
st_ymax

150506.126829327
```

## See Also

[ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMin](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 7.18.12 ST\_YMin

**ST\_YMin** — Returns the Y minima of a 2D or 3D bounding box or a geometry.

#### Synopsis

float **ST\_YMin**(box3d aGeomorBox2DorBox3D);

#### Description

Returns the Y minima of a 2D or 3D bounding box or a geometry.



#### Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



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



This method supports Circular Strings and Curves.

## Examples

```
SELECT ST_YMin('BOX3D(1 2 3, 4 5 6)');
st_ymin

2

SELECT ST_YMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_ymin

3

SELECT ST_YMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_ymin

2
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
a BOX3D
SELECT ST_YMin('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
150406 3)'));
st_ymin

150406
```

## See Also

[ST\\_GeomFromEWKT](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 7.18.13 ST\_ZMax

**ST\_ZMax** — Returns the Z maxima of a 2D or 3D bounding box or a geometry.

#### Synopsis

float **ST\_ZMax**(box3d aGeomorBox2DorBox3D);

#### Description

Returns the Z maxima of a 2D or 3D bounding box or a geometry.



#### Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



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



This method supports Circular Strings and Curves.

## Examples

```
SELECT ST_ZMax('BOX3D(1 2 3, 4 5 6)');
st_zmax

6

SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
st_zmax

7

SELECT ST_ZMax('BOX3D(-3 2 1, 3 4 1) ');
st_zmax

1
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
a BOX3D
SELECT ST_ZMax('LINESTRING(1 3 4, 5 6 7)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
150406 3)'));
st_zmax

3
```

## See Also

[ST\\_GeomFromEWKT](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#)

### 7.18.14 ST\_ZMin

**ST\_ZMin** — Returns the Z minima of a 2D or 3D bounding box or a geometry.

#### Synopsis

float **ST\_ZMin**(box3d aGeomorBox2DorBox3D);

#### Description

Returns the Z minima of a 2D or 3D bounding box or a geometry.



#### Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



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



This method supports Circular Strings and Curves.

## Examples

```
SELECT ST_ZMin('BOX3D(1 2 3, 4 5 6)');
st_zmin

3

SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
st_zmin

4

SELECT ST_ZMin('BOX3D(-3 2 1, 3 4 1) ');
st_zmin

1
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D
SELECT ST_ZMin('LINESTRING(1 3 4, 5 6 7)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'));
st_zmin

1
```

## See Also

[ST\\_GeomFromEWKT](#), [ST\\_GeomFromText](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#)

## 7.19 Referencia Lineal

### 7.19.1 ST\_LineInterpolatePoint

**ST\_LineInterpolatePoint** — Returns a point interpolated along a line at a fractional location.

#### Synopsis

geometry **ST\_LineInterpolatePoint**(geometry a\_linestring, float8 a\_fraction);  
 geography **ST\_LineInterpolatePoint**(geography a\_linestring, float8 a\_fraction, boolean use\_spheroid = true);

#### Descripción

Devuelve un punto interpolado a lo largo de una línea. El primer argumento debe ser un LINESTRING. El segundo argumento es un float8 entre 0 y 1 que representa la fracción de la longitud total de la cadena de línea del punto tiene que ser localizado.

Ver [ST\\_LineLocatePoint](#) para calcular la ubicación de la línea más cercana a un punto.



#### Note

This function computes points in 2D and then interpolates values for Z and M, while [ST\\_LineInterpolatePoint](#) computes points in 3D and only interpolates the M value.

**Note**

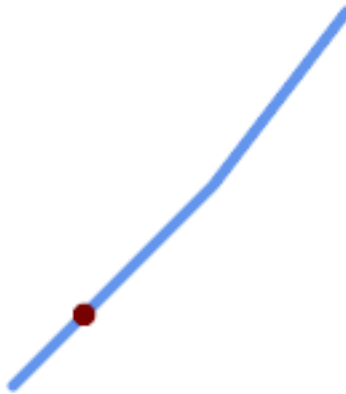
Desde la versión 1.1.1 esta función también interpola los valores M y Z (cuando están presentes), mientras que las versiones anteriores las establecen en 0.0.

Disponibilidad: 0.8.2, Z y M soportados añadidos en 1.1.1

Cambiado: 2.1.0. Hasta 2.0. x esto se llamaba ST\_Line\_Interpolate\_Point.



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

**Ejemplos**

*Una cadena de línea con el punto interpolado en la posición del 20% (0,20)*

```
-- The point 20% along a line

SELECT ST_AsEWKT(ST_LineInterpolatePoint(
 'LINESTRING(25 50, 100 125, 150 190)',
 0.2));

POINT(51.5974135047432 76.5974135047432)
```

The mid-point of a 3D line:

```
SELECT ST_AsEWKT(ST_LineInterpolatePoint('
 LINESTRING(1 2 3, 4 5 6, 6 7 8)',
 0.5));

POINT(3.5 4.5 5.5)
```

The closest point on a line to a point:

```
SELECT ST_AsText(ST_LineInterpolatePoint(line.geom,
 ST_LineLocatePoint(line.geom, 'POINT(4 3)'))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As geom) AS line;

POINT(3 4)
```



**Ver también**

[ST\\_LineInterpolatePoints](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineMerge](#)

**7.19.2 ST\_LineInterpolatePoint**

**ST\_LineInterpolatePoint** — Returns a point interpolated along a 3D line at a fractional location.

**Synopsis**

geometry **ST\_LineInterpolatePoint**(geometry a\_linestring, float8 a\_fraction);

**Descripción**

Devuelve un punto interpolado a lo largo de una línea. El primer argumento debe ser un **LINestring**. El segundo argumento es un float8 entre 0 y 1 que representa la fracción de la longitud total de la cadena de línea del punto tiene que ser localizado.

**Note**

**ST\_LineInterpolatePoint** computes points in 2D and then interpolates the values for Z and M, while this function computes points in 3D and only interpolates the M value.

Disponibilidad: 2.0.0



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

**Ejemplos**

Return point 20% along 3D line

```
SELECT ST_AsText (
 ST_3DLineInterpolatePoint('LINestring(25 50 70, 100 125 90, 150 190 200)',
 0.20));

 st_asetext

POINT Z (59.0675892910822 84.0675892910822 79.0846904776219)
```

**Ver también**

[ST\\_LineInterpolatePoint](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineMerge](#)

**7.19.3 ST\_LineInterpolatePoints**

**ST\_LineInterpolatePoints** — Returns points interpolated along a line at a fractional interval.

**Synopsis**

geometry **ST\_LineInterpolatePoints**(geometry a\_linestring, float8 a\_fraction, boolean repeat);  
 geography **ST\_LineInterpolatePoints**(geography a\_linestring, float8 a\_fraction, boolean use\_spheroid = true, boolean repeat = true);

## Descripción

Returns one or more points interpolated along a line at a fractional interval. The first argument must be a **LINESTRING**. The second argument is a float8 between 0 and 1 representing the spacing between the points as a fraction of line length. If the third argument is false, at most one point will be constructed (which is equivalent to **ST\_LineInterpolatePoint**.)

If the result has zero or one points, it is returned as a **POINT**. If it has two or more points, it is returned as a **MULTIPOINT**.

Availability: 2.5.0

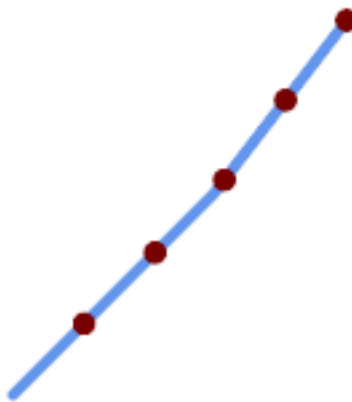


This function supports 3d and will not drop the z-index.



This function supports M coordinates.

## Ejemplos



*A LineString with points interpolated every 20%*

```
--Return points each 20% along a 2D line
SELECT ST_AsText(ST_LineInterpolatePoints('LINESTRING(25 50, 100 125, 150 190)', 0.20))

MULTIPOINT((51.5974135047432 76.5974135047432), (78.1948270094864 103.194827009486) ↵
, (104.132163186446 130.37181214238), (127.066081593223 160.18590607119), (150 190))
```

## Ver también

**ST\_LineInterpolatePoint**, **ST\_LineLocatePoint**

### 7.19.4 ST\_LineLocatePoint

**ST\_LineLocatePoint** — Returns the fractional location of the closest point on a line to a point.

## Synopsis

```
float8 ST_LineLocatePoint(geometry a_linestring, geometry a_point);
float8 ST_LineLocatePoint(geography a_linestring, geography a_point, boolean use_spheroid = true);
```

## Descripción

Devuelve un float entre 0 y 1 que representa la ubicación del punto más cercano en la cadena de línea al punto dado, como una fracción de la longitud total de la [2d line](#).

Puede utilizar la ubicación devuelta para extraer un punto ([ST\\_LineInterpolatePoint](#)) o una subcadena ([ST\\_LineSubstring](#)).

Esto es útil para aproximar números de direcciones

Disponibilidad: 1.1.0

Modificado: 2.1.0. Hasta 2.0.x esto se llamaba `ST_Line_Locate_Point`.

## Ejemplos

```
--Rough approximation of finding the street number of a point along the street
--Note the whole foo thing is just to generate dummy data that looks
--like house centroids and street
--We use ST_DWithin to exclude
--houses too far away from the street to be considered on the street
SELECT ST_AsText(house_loc) As as_text_house_loc,
 startstreet_num +
 CAST((endstreet_num - startstreet_num)
 * ST_LineLocatePoint(street_line, house_loc) As integer) As street_num
FROM
 (SELECT ST_GeomFromText('LINESTRING(1 2, 3 4)') As street_line,
 ST_Point(x*1.01,y*1.03) As house_loc, 10 As startstreet_num,
 20 As endstreet_num
 FROM generate_series(1,3) x CROSS JOIN generate_series(2,4) As y)
As foo
WHERE ST_DWithin(street_line, house_loc, 0.2);
```

| as_text_house_loc | street_num |
|-------------------|------------|
| POINT(1.01 2.06)  | 10         |
| POINT(2.02 3.09)  | 15         |
| POINT(3.03 4.12)  | 20         |

```
--find closest point on a line to a point or other geometry
SELECT ST_AsText(ST_LineInterpolatePoint(foo.the_line, ST_LineLocatePoint(foo.the_line,
 ST_GeomFromText('POINT(4 3)'))))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As the_line) As foo;
st_astext

POINT(3 4)
```

## Ver también

[ST\\_DWithin](#), [ST\\_Length2D](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineSubstring](#)

## 7.19.5 ST\_LineSubstring

`ST_LineSubstring` — Returns the part of a line between two fractional locations.

### Synopsis

geometry **ST\_LineSubstring**(geometry a\_linestring, float8 startfraction, float8 endfraction);  
 geography **ST\_LineSubstring**(geography a\_linestring, float8 startfraction, float8 endfraction);

## Descripción

Computes the line which is the section of the input line starting and ending at the given fractional locations. The first argument must be a LINESTRING. The second and third arguments are values in the range [0, 1] representing the start and end locations as fractions of line length. The Z and M values are interpolated for added endpoints if present.

Si 'Start' y 'End' tienen el mismo valor, esto equivale a **ST\_LineInterpolatePoint**.



### Note

This only works with LINESTRINGs. To use on contiguous MULTILINESTRINGs first join them with **ST\_LineMerge**.



### Note

Desde la versión 1.1.1 esta función también interpola los valores M y Z (cuando están presentes), mientras que las versiones anteriores las establecen en valores no especificados.

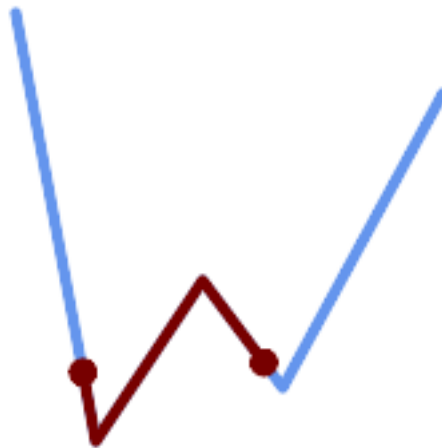
Disponibilidad: 1.1.0, Soporte de Z y M, añadido en 1.1.1

Modificado: 2.1.0. Hasta 2.0.x esto se llamaba ST\_Line\_Substring.



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

## Ejemplos



*Una cadena de línea vista con 1/3 de rango medio superpuestos (0,333, 0,666)*

```
SELECT ST_AsText(ST_LineSubstring('LINESTRING (20 180, 50 20, 90 80, 120 40, 180 150)',
0.333, 0.666));
```

```
LINESTRING (45.17311810399485 45.74337011202746, 50 20, 90 80, 112.97593050157862
49.36542599789519)
```

If start and end locations are the same, the result is a POINT.

```
SELECT ST_AsText(ST_LineSubstring('LINESTRING(25 50, 100 125, 150 190)', 0.333, 0.333));

POINT(69.2846934853974 94.2846934853974)
```

A query to cut a `LineString` into sections of length 100 or shorter. It uses `generate_series()` with a `CROSS JOIN LATERAL` to produce the equivalent of a `FOR` loop.

```
WITH data(id, geom) AS (VALUES
 ('A', 'LINESTRING(0 0, 200 0)::geometry),
 ('B', 'LINESTRING(0 100, 350 100)::geometry),
 ('C', 'LINESTRING(0 200, 50 200)::geometry)
)
SELECT id, i,
 ST_AsText(ST_LineSubstring(geom, startfrac, LEAST(endfrac, 1))) AS geom
FROM (
 SELECT id, geom, ST_Length(geom) len, 100 sublen FROM data
) AS d
CROSS JOIN LATERAL (
 SELECT i, (sublen * i) / len AS startfrac,
 (sublen * (i+1)) / len AS endfrac
 FROM generate_series(0, floor(len / sublen)::integer) AS t(i)
 -- skip last i if line length is exact multiple of sublen
 WHERE (sublen * i) / len <> 1.0
) AS d2;
```

| id | i | geom                        |
|----|---|-----------------------------|
| A  | 0 | LINESTRING(0 0,100 0)       |
| A  | 1 | LINESTRING(100 0,200 0)     |
| B  | 0 | LINESTRING(0 100,100 100)   |
| B  | 1 | LINESTRING(100 100,200 100) |
| B  | 2 | LINESTRING(200 100,300 100) |
| B  | 3 | LINESTRING(300 100,350 100) |
| C  | 0 | LINESTRING(0 200,50 200)    |

Ver también

[ST\\_Length](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineMerge](#)

7.19.6 ST\_LocateAlong

`ST_LocateAlong` — Returns the point(s) on a geometry that match a measure value.

Synopsis

geometry **ST\_LocateAlong**(geometry ageom\_with\_measure, float8 a\_measure, float8 offset);

Descripción

Returns the location(s) along a measured geometry that have the given measure values. The result is a `Point` or `MultiPoint`. Polygonal inputs are not supported.

If `offset` is provided, the result is offset to the left or right of the input line by the specified distance. A positive offset will be to the left, and a negative one to the right.

**Note**

Use this function only for linear geometries with an M component

The semantic is specified by the *ISO/IEC 13249-3 SQL/MM Spatial* standard.

Disponibilidad: 1.1.0 por antiguo nombre `ST_Locate_Along_Measure`.

Modificado: 2.0.0 en versiones anteriores éste solía llamarse `ST_Locate_Along_Measure`. El nombre anterior ha quedado obsoleto y se eliminará en el futuro, pero aún está disponible.



This function supports M coordinates.



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.13

**Ejemplos**

```
SELECT ST_AsText (
 ST_LocateAlong(
 'MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))'::geometry,
 3));

MULTIPOINT M ((1 2 3),(9 4 3),(1 2 3))
```

**Ver también**

[ST\\_LocateBetween](#), [ST\\_LocateBetweenElevations](#), [ST\\_InterpolatePoint](#)

**7.19.7 ST\_LocateBetween**

`ST_LocateBetween` — Returns the portions of a geometry that match a measure range.

**Synopsis**

geometry **ST\_LocateBetween**(geometry geomA, float8 measure\_start, float8 measure\_end, float8 offset);

**Descripción**

Devuelve un valor de la colección Geometry derivado con elementos que coinciden con la medida especificada. No se admiten elementos poligonales.

Si se proporciona un desplazamiento, el resultado se desplazará a la izquierda o a la derecha de la línea de entrada por el número de unidades especificado. Un desplazamiento positivo será a la izquierda, y uno negativo a la derecha.

Clipping a non-convex POLYGON may produce invalid geometry.

The semantic is specified by the *ISO/IEC 13249-3 SQL/MM Spatial* standard.

Disponibilidad: 1.1.0 por nombre antiguo `ST_Locate_Between_Measures`.

Modificado: 2.0.0 en versiones anteriores éste solía llamarse `ST_Locate_Along_Measure`. El nombre anterior ha quedado obsoleto y se eliminará en el futuro, pero aún está disponible.

Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE.



This function supports M coordinates.

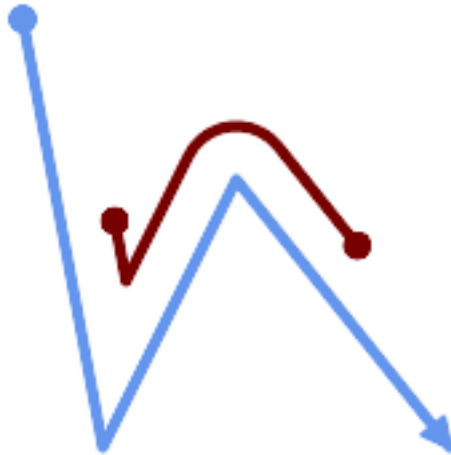


This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

## Ejemplos

```
SELECT ST_AsText(
 ST_LocateBetween(
 'MULTILINESTRING M ((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))':: geometry,
 1.5, 3));

GEOMETRYCOLLECTION M (LINESTRING M (1 2 3,3 4 2,9 4 3),POINT M (1 2 3))
```



*A LineString with the section between measures 2 and 8, offset to the left*

```
SELECT ST_AsText(ST_LocateBetween(
 ST_AddMeasure('LINESTRING (20 180, 50 20, 100 120, 180 20)', 0, 10),
 2, 8,
 20
));

MULTILINESTRING((54.49835019899045 104.53426957938231,58.70056060327303 ↵
 82.12248075654186,69.16695286779743 103.05526528559065,82.11145618000168 ↵
 128.94427190999915,84.24893681714357 132.32493442618113,87.01636951231555 ↵
 135.21267035596549,90.30307285299679 137.49198684843182,93.97759758337769 ↵
 139.07172433557758,97.89298381958797 139.8887023914453,101.89263860095893 ↵
 139.9102465862721,105.81659870902816 139.13549527600819,109.50792827749828 ↵
 137.5954340631298,112.81899532549731 135.351656550512,115.6173761888606 ↵
 132.49390095108848,145.31017306064817 95.37790486135405))
```

## Ver también

[ST\\_LocateAlong](#), [ST\\_LocateAlong](#), [ST\\_LocateBetween](#)

## 7.19.8 ST\_LocateBetweenElevations

**ST\_LocateBetweenElevations** — Returns the portions of a geometry that lie in an elevation (Z) range.

### Synopsis

geometry **ST\_LocateBetweenElevations**(geometry geom\_mline, float8 elevation\_start, float8 elevation\_end);

## Descripción

Returns a geometry (collection) with the portions of a geometry that lie in an elevation (Z) range.

Clipping a non-convex POLYGON may produce invalid geometry.

Disponibilidad: 1.4.0

Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE.



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

## Ejemplos

```
SELECT ST_AsText (
 ST_LocateBetweenElevations(
 'LINESTRING(1 2 3, 4 5 6)::geometry,
 2, 4));

 st_astext

MULTILINESTRING Z ((1 2 3,2 3 4))

SELECT ST_AsText (
 ST_LocateBetweenElevations(
 'LINESTRING(1 2 6, 4 5 -1, 7 8 9)',
 6, 9)) As ewelev;

 ewelev

GEOMETRYCOLLECTION Z (POINT Z (1 2 6),LINESTRING Z (6.1 7.1 6,7 8 9))
```

## Ver también

[ST\\_Dump](#), [ST\\_LocateAlong](#), [ST\\_LocateBetween](#)

## 7.19.9 ST\_InterpolatePoint

ST\_InterpolatePoint — Devuelve el valor de la dimensión medida de una geometría en el punto cerrado al punto proporcionado.

## Synopsis

float8 **ST\_InterpolatePoint**(geometry linear\_geom\_with\_measure, geometry point);

## Descripción

Returns an interpolated measure value of a linear measured geometry at the location closest to the given point.



### Note

Use this function only for linear geometries with an M component

Disponibilidad: 2.0.0



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



## Ejemplos

```
SELECT ST_InterpolatePoint('LINESTRING M (0 0 0, 10 0 20)', 'POINT(5 5)');

10
```

## Ver también

[ST\\_AddMeasure](#), [ST\\_LocateAlong](#), [ST\\_LocateBetween](#)

### 7.19.10 ST\_AddMeasure

ST\_AddMeasure — Interpolates measures along a linear geometry.

## Synopsis

geometry **ST\_AddMeasure**(geometry geom\_mline, float8 measure\_start, float8 measure\_end);

## Descripción

Devuelve una geometría derivada con elementos de medida interpolados linealmente entre los puntos inicial y final. Si la geometría no tiene una dimensión de medida, se añadirá una. Si la geometría tiene una dimensión de medida, se sobrescribe con nuevos valores. Sólo se admiten LINESTRINGS y MULTILINESTRINGS.

Disponibilidad: 1.5.0



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

## Ejemplos

```
SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0, 2 0, 4 0)'),1,4)) As ewelev;

LINESTRINGM(1 0 1,2 0 2,4 0 4)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0 4,2 0 4, 4 0 4)'),10,40)) As ewelev;

LINESTRING(1 0 4 10,2 0 4 20,4 0 4 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRINGM(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;

LINESTRINGM(1 0 10,2 0 20,4 0 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('MULTILINESTRINGM((1 0 4, 2 0 4, 4 0 4),(1 0 4, 2 0 4, 4 0 4)'),10,70)) As ←
ewelev;

MULTILINESTRINGM((1 0 10,2 0 20,4 0 40),(1 0 40,2 0 50,4 0 70))
```

## 7.20 Trajectory Functions

### 7.20.1 ST\_IsValidTrajectory

**ST\_IsValidTrajectory** — Tests if the geometry is a valid trajectory.

#### Synopsis

boolean **ST\_IsValidTrajectory**(geometry line);

#### Description

Tests if a geometry encodes a valid trajectory. A valid trajectory is represented as a `LINESTRING` with measures (M values). The measure values must increase from each vertex to the next.

Valid trajectories are expected as input to spatio-temporal functions like [ST\\_ClosestPointOfApproach](#)

Availability: 2.2.0



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

#### Examples

```
-- A valid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(
 ST_MakePointM(0,0,1),
 ST_MakePointM(0,1,2))
);
t

-- An invalid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0,0,1), ST_MakePointM(0,1,0)));
NOTICE: Measure of vertex 1 (0) not bigger than measure of vertex 0 (1)
st_isvalidtrajectory

f
```

#### See Also

[ST\\_ClosestPointOfApproach](#)

### 7.20.2 ST\_ClosestPointOfApproach

**ST\_ClosestPointOfApproach** — Returns a measure at the closest point of approach of two trajectories.

#### Synopsis

float8 **ST\_ClosestPointOfApproach**(geometry track1, geometry track2);

Description

Returns the smallest measure at which points interpolated along the given trajectories are at the smallest distance.  
Inputs must be valid trajectories as checked by [ST\\_IsValidTrajectory](#). Null is returned if the trajectories do not overlap in their M ranges.

See [ST\\_LocateAlong](#) for getting the actual points at the given measure.

Availability: 2.2.0

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

Examples

```
-- Return the time in which two objects moving between 10:00 and 11:00
-- are closest to each other and their distance at that point
WITH inp AS (SELECT
 ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5) '::geometry,
 extract(epoch from '2015-05-26 10:00'::timestampz),
 extract(epoch from '2015-05-26 11:00'::timestampz)
) a,
 ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2) '::geometry,
 extract(epoch from '2015-05-26 10:00'::timestampz),
 extract(epoch from '2015-05-26 11:00'::timestampz)
) b
), cpa AS (
 SELECT ST_ClosestPointOfApproach(a,b) m FROM inp
), points AS (
 SELECT ST_Force3DZ(ST_GeometryN(ST_LocateAlong(a,m),1)) pa,
 ST_Force3DZ(ST_GeometryN(ST_LocateAlong(b,m),1)) pb
 FROM inp, cpa
)
SELECT to_timestamp(m) t,
 ST_Distance(pa,pb) distance
FROM points, cpa;
```

| t                             | distance         |
|-------------------------------|------------------|
| 2015-05-26 10:45:31.034483+02 | 1.96036833151395 |

See Also

[ST\\_IsValidTrajectory](#), [ST\\_DistanceCPA](#), [ST\\_LocateAlong](#), [ST\\_AddMeasure](#)

7.20.3 ST\_DistanceCPA

ST\_DistanceCPA — Returns the distance between the closest point of approach of two trajectories.

Synopsis

float8 **ST\_DistanceCPA**(geometry track1, geometry track2);

**Description**

Returns the minimum distance two moving objects have ever been each other.

Inputs must be valid trajectories as checked by [ST\\_IsValidTrajectory](#). Null is returned if the trajectories do not overlap in their M ranges.

Availability: 2.2.0



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

**Examples**

```
-- Return the minimum distance of two objects moving between 10:00 and 11:00
WITH inp AS (SELECT
 ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5) '::geometry,
 extract(epoch from '2015-05-26 10:00'::timestampz),
 extract(epoch from '2015-05-26 11:00'::timestampz)
) a,
 ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2) '::geometry,
 extract(epoch from '2015-05-26 10:00'::timestampz),
 extract(epoch from '2015-05-26 11:00'::timestampz)
) b
)
SELECT ST_DistanceCPA(a,b) distance FROM inp;

 distance

1.96036833151395
```

**See Also**

[ST\\_IsValidTrajectory](#), [ST\\_ClosestPointOfApproach](#), [ST\\_AddMeasure](#), [ST\\_DistanceCPA](#)

**7.20.4 ST\_CPAWithin**

**ST\_CPAWithin** — Tests if the closest point of approach of two trajectories is within the specified distance.

**Synopsis**

boolean **ST\_CPAWithin**(geometry track1, geometry track2, float8 dist);

**Description**

Tests whether two moving objects have ever been closer than the specified distance.

Inputs must be valid trajectories as checked by [ST\\_IsValidTrajectory](#). False is returned if the trajectories do not overlap in their M ranges.

Availability: 2.2.0



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

Examples

```
WITH inp AS (SELECT
 ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)'::geometry,
 extract(epoch from '2015-05-26 10:00'::timestampz),
 extract(epoch from '2015-05-26 11:00'::timestampz)
) a,
 ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)'::geometry,
 extract(epoch from '2015-05-26 10:00'::timestampz),
 extract(epoch from '2015-05-26 11:00'::timestampz)
) b
)
SELECT ST_CPAWithin(a,b,2), ST_DistanceCPA(a,b) distance FROM inp;
```

| st_cpawithin | distance         |
|--------------|------------------|
| t            | 1.96521473776207 |

See Also

[ST\\_IsValidTrajectory](#), [ST\\_ClosestPointOfApproach](#), [ST\\_DistanceCPA](#), [|](#)

7.21 SFCGAL Functions

7.21.1 postgis\_sfcgal\_version

postgis\_sfcgal\_version — Returns the version of SFCGAL in use

Synopsis

text **postgis\_sfcgal\_version**(void);

Descripción

Returns the version of SFCGAL in use

Disponibilidad: 2.1.0

-  This method needs SFCGAL backend.
-  This function supports 3d and will not drop the z-index.
-  This function supports Polyhedral surfaces.
-  This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Ver también

[postgis\\_sfcgal\\_full\\_version](#)

7.21.2 postgis\_sfcgal\_full\_version

postgis\_sfcgal\_full\_version — Returns the full version of SFCGAL in use including CGAL and Boost versions

## Synopsis

text **postgis\_sfcgal\_full\_version**(void);

## Descripción

Returns the full version of SFCGAL in use including CGAL and Boost versions

Disponibilidad: 2.1.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Ver también

[postgis\\_sfcgal\\_version](#)

## 7.21.3 ST\_3DArea

ST\_3DArea — Computes area of 3D surface geometries. Will return 0 for solids.

## Synopsis

float**ST\_3DArea**(geometry geom1);

## Descripción

Disponibilidad: 2.1.0



This method needs SFCGAL backend.



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 8.1, 10.5



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



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Ejemplos

Note: By default a PolyhedralSurface built from WKT is a surface geometry, not solid. It therefore has surface area. Once converted to a solid, no area.

```
SELECT ST_3DArea(geom) As cube_surface_area,
 ST_3DArea(ST_MakeSolid(geom)) As solid_surface_area
FROM (SELECT 'POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
 ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
 ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
 ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
 ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
 ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))'::geometry) As f(geom);

cube_surface_area | solid_surface_area
-----+-----
6 | 0
```

Ver también

[ST\\_Area](#), [ST\\_MakeSolid](#), [ST\\_IsSolid](#), [ST\\_Area](#)

7.21.4 ST\_3DConvexHull

ST\_3DConvexHull — Computes the 3D convex hull of a geometry.

Synopsis

geometry **ST\_3DConvexHull**(geometry geom1);

Descripción

Disponibilidad: 2.1.0

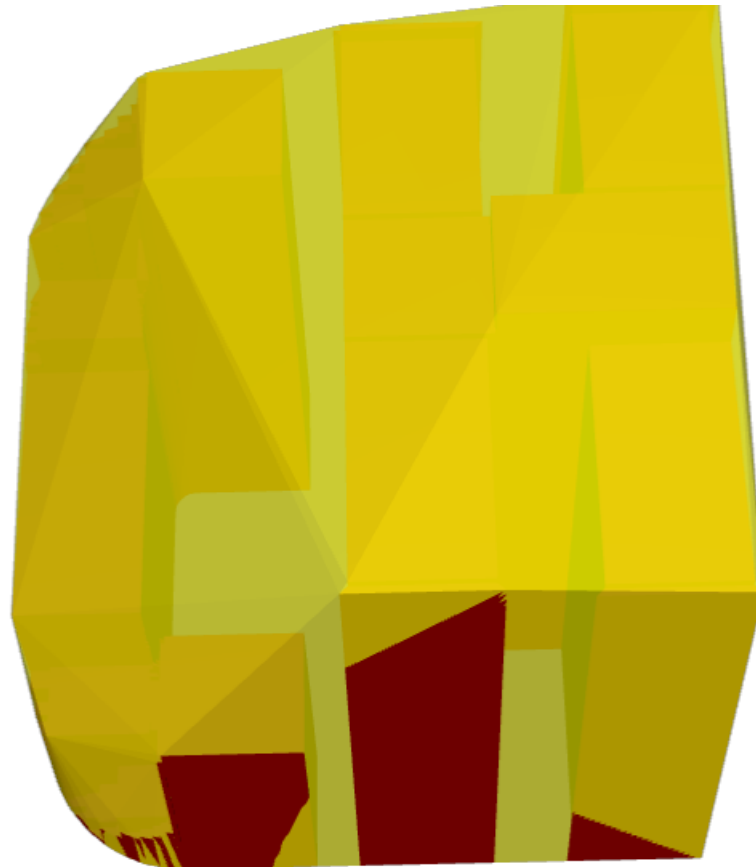
-  This method needs SFCGAL backend.
-  This function supports 3d and will not drop the z-index.
-  This function supports Polyhedral surfaces.
-  This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Ejemplos

```
SELECT ST_AsText(ST_3DConvexHull('LINESTRING Z(0 0 5, 1 5 3, 5 7 6, 9 5 3 , 5 7 5, 6 3 5) '↵
'::geometry));

POLYHEDRALSURFACE Z (((1 5 3,9 5 3,0 0 5,1 5 3)),((1 5 3,0 0 5,5 7 6,1 5 3)),((5 7 6,5 7 ↵
5,1 5 3,5 7 6)),((0 0 5,6 3 5,5 7 6,0 0 5)),((6 3 5,9 5 3,5 7 6,6 3 5)),((0 0 5,9 5 3,6 ↵
3 5,0 0 5)),((9 5 3,5 7 5,5 7 6,9 5 3)),((1 5 3,5 7 5,9 5 3,1 5 3)))

WITH f AS (SELECT i, ST_Extrude(geom, 0,0, i) AS geom
FROM ST_Subdivide(ST_Letters('CH'),5) WITH ORDINALITY AS sd(geom,i)
)
SELECT ST_3DConvexHull(ST_Collect(f.geom))
FROM f;
```



*Original geometry overlaid with 3D convex hull*

**Ver también**

[ST\\_Letters](#), [ST\\_AsX3D](#)

### 7.21.5 ST\_3DIntersection

ST\_3DIntersection — Perform 3D intersection

#### Synopsis

```
geometry ST_3DIntersection(geometry geom1, geometry geom2);
```

#### Descripción

Return a geometry that is the shared portion between geom1 and geom2.

Disponibilidad: 2.1.0

- ✓ This method needs SFCGAL backend.
- ✓ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- ✓ This function supports 3d and will not drop the z-index.

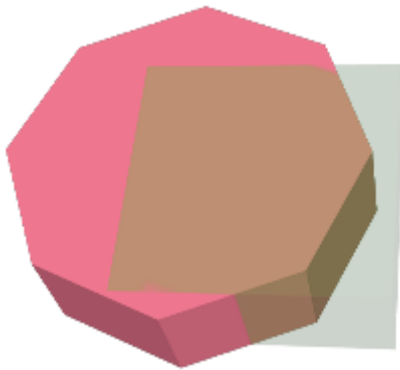


- ✓ This function supports Polyhedral surfaces.
- ✓ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Ejemplos

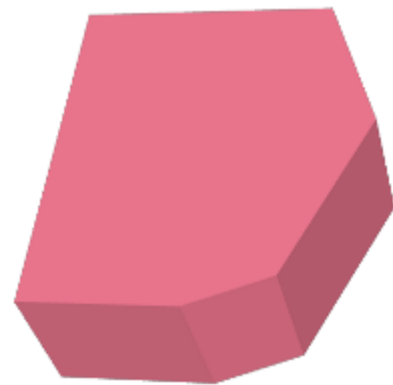
3D images were generated using PostGIS [ST\\_AsX3D](#) and rendering in HTML using [X3Dom HTML Javascript rendering library](#).

```
SELECT ST_Extrude(ST_Buffer(↵
 ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
 ST_Extrude(ST_Buffer(↵
 ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2;
```



*Original 3D geometries overlaid. geom2 is shown semi-transparent*

```
SELECT ST_3DIntersection(geom1,geom2)
FROM (SELECT ST_Extrude(ST_Buffer(↵
 ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
 ST_Extrude(ST_Buffer(↵
 ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2) As ↵
t;
```



*Intersection of geom1 and geom2*

## 3D linestrings and polygons

```
SELECT ST_AsText(ST_3DIntersection(linestring, polygon)) As wkt
FROM ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS ↵
linestring
CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;
```

wkt

```

LINESTRING Z (1 1 8,0.5 0.5 8)
```

## Cube (closed Polyhedral Surface) and Polygon Z

```
SELECT ST_AsText(ST_3DIntersection(
 ST_GeomFromText('POLYHEDRALSURFACE Z(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)) ↵
 ,
 ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
 ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
 ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))'),
 'POLYGON Z ((0 0 0, 0 0 0.5, 0 0.5 0.5, 0 0.5 0, 0 0 0))'::geometry))
```

```
TIN Z (((0 0 0,0 0 0.5,0 0.5 0.5,0 0 0)),((0 0.5 0,0 0 0,0 0.5 0.5,0 0.5 0)))
```

Intersection of 2 solids that result in volumetric intersection is also a solid (ST\_Dimension returns 3)

```
SELECT ST_AsText(ST_3DIntersection(ST_Extrude(ST_Buffer('POINT(10 20) '::geometry,10,1) ←
,0,0,30),
ST_Extrude(ST_Buffer('POINT(10 20) '::geometry,10,1),2,0,10)));

POLYHEDRALSURFACE Z (((13.333333333333 13.333333333333 10,20 20 0,20 20 ←
10,13.333333333333 13.333333333333 10)),
((20 20 10,16.666666666667 23.333333333333 10,13.333333333333 13.333333333333 ←
10,20 20 10)),
((20 20 0,16.666666666667 23.333333333333 10,20 20 10,20 20 0)),
((13.333333333333 13.333333333333 10,10 10 0,20 20 0,13.333333333333 ←
13.333333333333 10)),
((16.666666666667 23.333333333333 10,12 28 10,13.333333333333 13.333333333333 ←
10,16.666666666667 23.333333333333 10)),
((20 20 0,9.9999999999995 30 0,16.666666666667 23.333333333333 10,20 20 0)),
((10 10 0,9.9999999999995 30 0,20 20 0,10 10 0)),((13.333333333333 ←
13.333333333333 10,12 12 10,10 10 0,13.333333333333 13.333333333333 10)),
((12 28 10,12 12 10,13.333333333333 13.333333333333 10,12 28 10)),
((16.666666666667 23.333333333333 10,9.9999999999995 30 0,12 28 ←
10,16.666666666667 23.333333333333 10)),
((10 10 0,0 20 0,9.9999999999995 30 0,10 10 0)),
((12 12 10,11 11 10,10 10 0,12 12 10)),((12 28 10,11 11 10,12 12 10,12 28 10)),
((9.9999999999995 30 0,11 29 10,12 28 10,9.9999999999995 30 0)),((0 20 0,2 20 ←
10,9.9999999999995 30 0,0 20 0)),
((10 10 0,2 20 10,0 20 0,10 10 0)),((11 11 10,2 20 10,10 10 0,11 11 10)),((12 28 ←
10,11 29 10,11 11 10,12 28 10)),
((9.9999999999995 30 0,2 20 10,11 29 10,9.9999999999995 30 0)),((11 11 10,11 29 ←
10,2 20 10,11 11 10)))
```

## 7.21.6 ST\_3DDifference

ST\_3DDifference — Perform 3D difference

### Synopsis

geometry **ST\_3DDifference**(geometry geom1, geometry geom2);

### Descripción

Returns that part of geom1 that is not part of geom2.

Disponibilidad: 2.2.0



This method needs SFCGAL backend.



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1



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



This function supports Polyhedral surfaces.

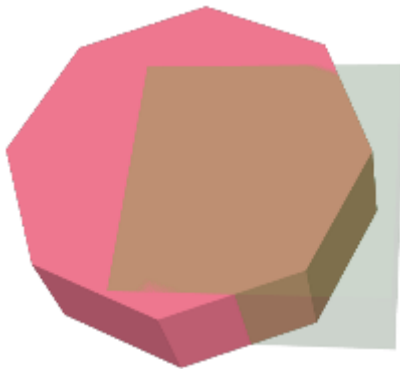


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### Ejemplos

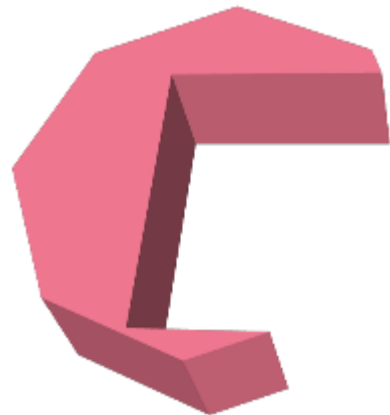
3D images were generated using PostGIS **ST\_AsX3D** and rendering in HTML using **X3Dom HTML Javascript rendering library**.

```
SELECT ST_Extrude(ST_Buffer(↵
 ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
 ST_Extrude(ST_Buffer(↵
 ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2;
```



*Original 3D geometries overlaid. geom2 is the part that will be removed.*

```
SELECT ST_3DDifference(geom1,geom2)
FROM (SELECT ST_Extrude(ST_Buffer(↵
 ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
 ST_Extrude(ST_Buffer(↵
 ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2) As ↵
t;
```



*What's left after removing geom2*

#### Ver también

[ST\\_Extrude](#), [ST\\_AsX3D](#), [ST\\_3DIntersection](#) [ST\\_3DUnion](#)

### 7.21.7 ST\_3DUnion

ST\_3DUnion — Perform 3D union.

#### Synopsis

geometry **ST\_3DUnion**(geometry geom1, geometry geom2);  
 geometry **ST\_3DUnion**(geometry set g1field);

#### Descripción

Disponibilidad: 2.2.0

Availability: 3.3.0 aggregate variant was added



This method needs SFCGAL backend.



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1



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



This function supports Polyhedral surfaces.

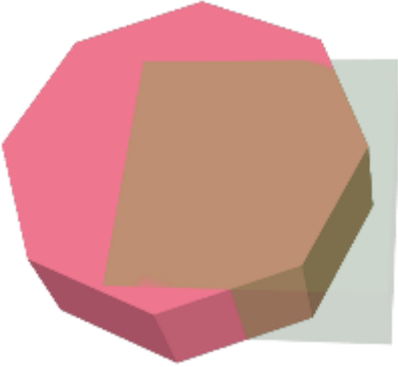
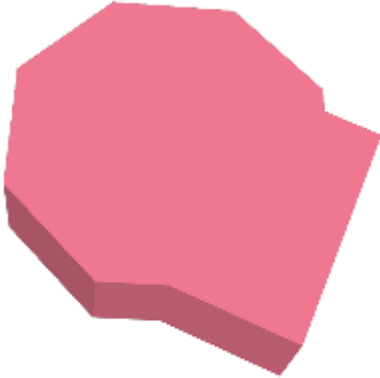


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**Aggregate variant:** returns a geometry that is the 3D union of a rowset of geometries. The `ST_3DUnion()` function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the `SUM()` and `AVG()` functions do and like most aggregates, it also ignores NULL geometries.

Ejemplos

3D images were generated using PostGIS `ST_AsX3D` and rendering in HTML using `X3Dom HTML Javascript rendering library`.

|                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>SELECT ST_Extrude(ST_Buffer( ←<br/>    ST_GeomFromText('POINT(100 90)'),<br/>    50, 'quad_segs=2'),0,0,30) AS geom1,<br/>    ST_Extrude(ST_Buffer( ←<br/>    ST_GeomFromText('POINT(80 80)'),<br/>    50, 'quad_segs=1'),0,0,30) AS geom2;</pre>  <p><i>Original 3D geometries overlaid. geom2 is the one with transparency.</i></p> | <pre>SELECT ST_3DUnion(geom1,geom2)<br/>FROM ( SELECT ST_Extrude(ST_Buffer( ←<br/>    ST_GeomFromText('POINT(100 90)'),<br/>    50, 'quad_segs=2'),0,0,30) AS geom1,<br/>    ST_Extrude(ST_Buffer( ←<br/>    ST_GeomFromText('POINT(80 80)'),<br/>    50, 'quad_segs=1'),0,0,30) AS geom2 ) As ←<br/>t;</pre>  <p><i>Union of geom1 and geom2</i></p> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Ver también

`ST_Extrude`, `ST_AsX3D`, `ST_3DIntersection` `ST_3DDifference`

7.21.8 ST\_AlphaShape

`ST_AlphaShape` — Computes an Alpha-shape enclosing a geometry

Synopsis

geometry `ST_AlphaShape`(geometry geom, float alpha, boolean allow\_holes = false);

Descripción

Computes the **Alpha-Shape** of the points in a geometry. An alpha-shape is a (usually) concave polygonal geometry which contains all the vertices of the input, and whose vertices are a subset of the input vertices. An alpha-shape provides a closer fit to the shape of the input than the shape produced by the **convex hull**.

The "closeness of fit" is controlled by the `alpha` parameter, which can have values from 0 to infinity. Smaller alpha values produce more concave results. Alpha values greater than some data-dependent value produce the convex hull of the input.



#### Note

Following the CGAL implementation, the alpha value is the *square* of the radius of the disc used in the Alpha-Shape algorithm to "erode" the Delaunay Triangulation of the input points. See [CGAL Alpha-Shapes](#) for more information. This is different from the original definition of alpha-shapes, which defines alpha as the radius of the eroding disc.

The computed shape does not contain holes unless the optional `allow_holes` argument is specified as true.

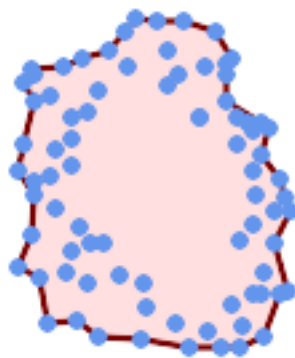
This function effectively computes a concave hull of a geometry in a similar way to `ST_ConcaveHull`, but uses CGAL and a different algorithm.

Availability: 3.3.0 - requires SFCGAL >= 1.4.1.



This method needs SFCGAL backend.

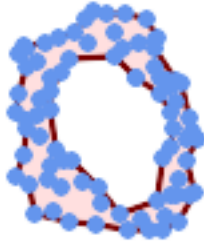
## Ejemplos



*Alpha-shape of a MultiPoint (same example As [ST\\_OptimalAlphaShape](#))*

```
SELECT ST_AsText(ST_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),
(36 61),(32 65),
(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),
(27 84),(52 98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),
(27 77),(39 88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),
(69 86),(60 90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),
(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry,80.2));
```

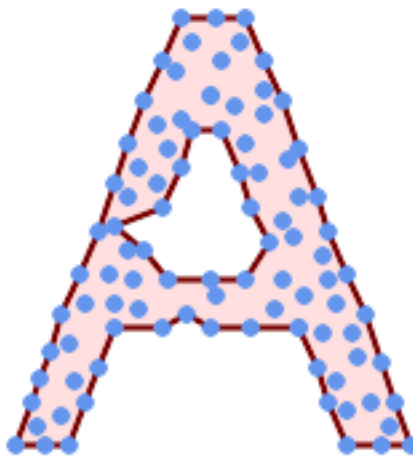
```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 33,
23 36,26 44,27 54,23 60,24 67,
27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,64 97,72 95,76 88,75 84,83 72,
85 71,88 58,89 53));
```



Alpha-shape of a MultiPoint, allowing holes (same example as *ST\_OptimalAlphaShape*)

```
SELECT ST_AsText(ST_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70) ↵
, (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61) ↵
, (32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
 (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ↵
 29),(27 84),(52 98),(72 95),(85 71),
 (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ↵
 97),(27 77),(39 88),(60 81),
 (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ↵
 64),(69 86),(60 90),(50 86),(43 80),(36 73),
 (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ↵
 16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry, 100.1,true))

POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ↵
33,23 36,26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 ↵
97,64 97,72 95,76 88,75 84,83 72,85 71,88 58,89 53),
 (36 61,36 68,40 75,43 80,50 86,60 81,68 73,77 67,81 60,82 54,81 47,78 43,76 ↵
 27,62 22,54 32,48 34,44 42,38 46,36 61))
```



Alpha-shape of a MultiPoint, allowing holes (same example as *ST\_ConcaveHull*)

```

SELECT ST_AlphaShape(
 'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 36), (46 20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 100), (63 118), (68 133), (74 149), (81 164), (88 180), (101 180), (112 180), (119 164), (126 149), (132 131), (139 113), (143 100), (150 84), (157 69), (163 51), (168 36), (174 20), (163 20), (150 20), (143 36), (139 49), (132 64), (99 151), (92 138), (88 124), (81 109), (74 93), (70 82), (83 82), (99 82), (112 82), (126 82), (121 96), (114 109), (110 122), (103 138), (99 151), (34 27), (43 31), (48 44), (46 58), (52 73), (63 73), (61 84), (72 71), (90 69), (101 76), (123 71), (141 62), (166 27), (150 33), (159 36), (146 44), (154 53), (152 62), (146 73), (134 76), (143 82), (141 91), (130 98), (126 104), (132 113), (128 127), (117 122), (112 133), (119 144), (108 147), (119 153), (110 171), (103 164), (92 171), (86 160), (88 142), (79 140), (72 124), (83 131), (79 118), (68 113), (63 102), (68 93), (35 45))'::geometry,102.2, true);

POLYGON((134 80,136 75,130 63,135 45,132 44,126 28,117 24,110 24,98 24,80 27,82 39,72 51,60 48,56 34,52 52,42 50,
34 54,39 66,40 81,34 90,36 100,40 116,36 123,39 128,51 129,58 132,68 135,74 142,78 147,86 146,96 146,
108 142,114 132,112 126,112 116,116 110,120 108,125 108,128 106,125 96,132 87,134 80))

```

## Ver también

[ST\\_ConcaveHull](#), [ST\\_OptimalAlphaShape](#)

## 7.21.9 ST\_ApproximateMedialAxis

**ST\_ApproximateMedialAxis** — Compute the approximate medial axis of an areal geometry.

### Synopsis

geometry **ST\_ApproximateMedialAxis**(geometry geom);

### Descripción

Return an approximate medial axis for the areal input based on its straight skeleton. Uses an SFCGAL specific API when built against a capable version (1.2.0+). Otherwise the function is just a wrapper around `ST_StraightSkeleton` (slower case).

Disponibilidad: 2.2.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



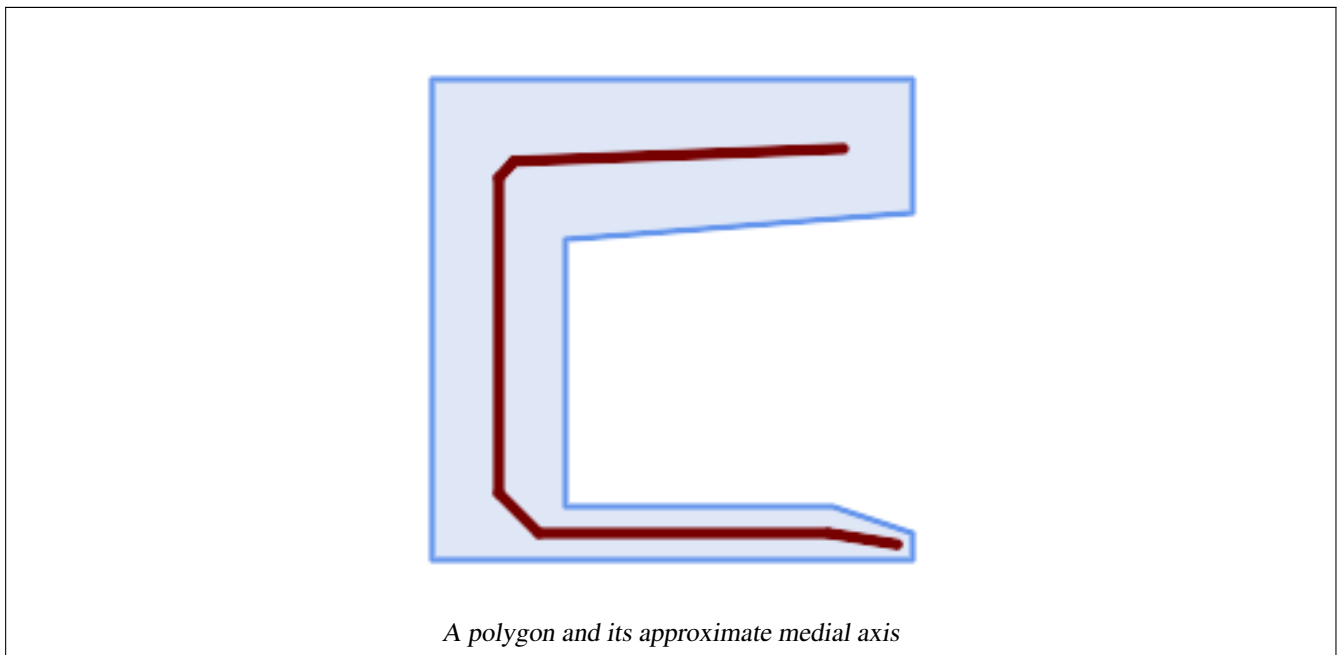
This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### Ejemplos

```

SELECT ST_ApproximateMedialAxis(ST_GeomFromText('POLYGON ((190 190, 10 190, 10 10, 190 10, 190 20, 160 30, 60 30, 60 130, 190 140, 190 190))'));

```



Ver también

[ST\\_StraightSkeleton](#)

### 7.21.10 ST\_ConstrainedDelaunayTriangles

ST\_ConstrainedDelaunayTriangles — Return a constrained Delaunay triangulation around the given input geometry.

#### Synopsis

```
geometry ST_ConstrainedDelaunayTriangles(geometry g1);
```

#### Descripción

Return a **Constrained Delaunay triangulation** around the vertices of the input geometry. Output is a TIN.



This method needs SFCGAL backend.

Disponibilidad: 2.1.0

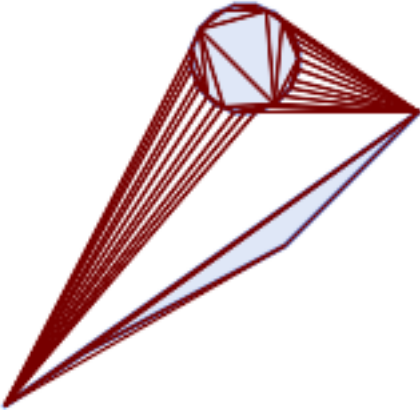
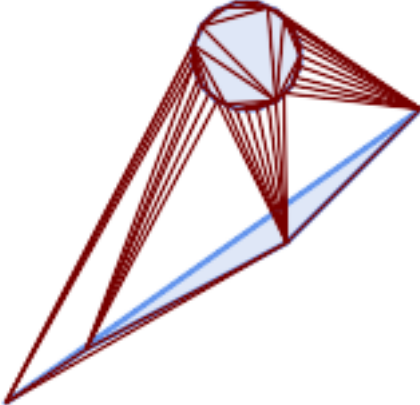


This function supports 3d and will not drop the z-index.

#### Ejemplos

---



|                                                                                                                                                                                                                                                                                                                                                                                        |                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  <p><i>ST_ConstrainedDelaunayTriangles</i> of 2 polygons</p> <pre>select ST_ConstrainedDelaunayTriangles(<br/>    ST_Union(<br/>        'POLYGON((175 150, ↵<br/>    20 40, 50 60, 125 100, 175 150))'::geometry,<br/>        ST_Buffer('POINT ↵<br/>(110 170)'::geometry, 20)<br/>    )<br/>);</pre> |  <p><i>ST_DelaunayTriangles</i> of 2 polygons. Triangle edges cross polygon boundaries.</p> <pre>select ST_DelaunayTriangles(<br/>    ST_Union(<br/>        'POLYGON((175 150, ↵<br/>    20 40, 50 60, 125 100, 175 150))'::geometry,<br/>        ST_Buffer('POINT ↵<br/>(110 170)'::geometry, 20)<br/>    )<br/>);</pre> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Ver también**

[ST\\_DelaunayTriangles](#), [ST\\_TriangulatePolygon](#), [ST\\_Tesselate](#), [ST\\_ConcaveHull](#), [ST\\_Dump](#)

**7.21.11 ST\_Extrude**

ST\_Extrude — Extrude a surface to a related volume

**Synopsis**

geometry **ST\_Extrude**(geometry geom, float x, float y, float z);

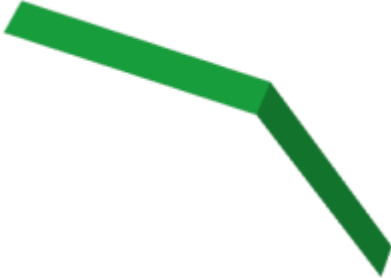
**Descripción**

Disponibilidad: 2.1.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Ejemplos

3D images were generated using PostGIS **ST\_AsX3D** and rendering in HTML using **X3Dom HTML Javascript rendering library**.

|                                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>SELECT ST_Buffer(ST_GeomFromText('POINT ↵ (100 90)'), 50, 'quad_segs=2'),0,0,30);</pre>  <p><i>Original octagon formed from buffering point</i></p> | <pre>ST_Extrude(ST_Buffer(ST_GeomFromText(' ↵ POINT(100 90)'), 50, 'quad_segs=2'),0,0,30);</pre>  <p><i>Hexagon extruded 30 units along Z produces a PolyhedralSurfaceZ</i></p> |
| <pre>SELECT ST_GeomFromText('LINESTRING(50 50, ↵ 100 90, 95 150)')</pre>  <p><i>Original linestring</i></p>                                            | <pre>SELECT ST_Extrude( ST_GeomFromText('LINESTRING(50 50, 100 ↵ 90, 95 150)'),0,0,10));</pre>  <p><i>LineString Extruded along Z produces a PolyhedralSurfaceZ</i></p>       |

Ver también

**ST\_AsX3D**

### 7.21.12 ST\_ForceLHR

ST\_ForceLHR — Force LHR orientation

#### Synopsis

geometry **ST\_ForceLHR**(geometry geom);

#### Descripción

Disponibilidad: 2.1.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### 7.21.13 ST\_IsPlanar

ST\_IsPlanar — Check if a surface is or not planar

#### Synopsis

boolean **ST\_IsPlanar**(geometry geom);

#### Descripción

Availability: 2.2.0: This was documented in 2.1.0 but got accidentally left out in 2.1 release.



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### 7.21.14 ST\_IsSolid

ST\_IsSolid — Test if the geometry is a solid. No validity check is performed.

#### Synopsis

boolean **ST\_IsSolid**(geometry geom1);

---

### Descripción

Disponibilidad: 2.2.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### 7.21.15 ST\_MakeSolid

**ST\_MakeSolid** — Cast the geometry into a solid. No check is performed. To obtain a valid solid, the input geometry must be a closed Polyhedral Surface or a closed TIN.

### Synopsis

geometry **ST\_MakeSolid**(geometry geom1);

### Descripción

Disponibilidad: 2.2.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### 7.21.16 ST\_MinkowskiSum

**ST\_MinkowskiSum** — Performs Minkowski sum

### Synopsis

geometry **ST\_MinkowskiSum**(geometry geom1, geometry geom2);

### Descripción

This function performs a 2D minkowski sum of a point, line or polygon with a polygon.

A minkowski sum of two geometries A and B is the set of all points that are the sum of any point in A and B. Minkowski sums are often used in motion planning and computer-aided design. More details on [Wikipedia Minkowski addition](#).

The first parameter can be any 2D geometry (point, linestring, polygon). If a 3D geometry is passed, it will be converted to 2D by forcing Z to 0, leading to possible cases of invalidity. The second parameter must be a 2D polygon.

Implementation utilizes [CGAL 2D Minkowskisum](#).

Disponibilidad: 2.1.0

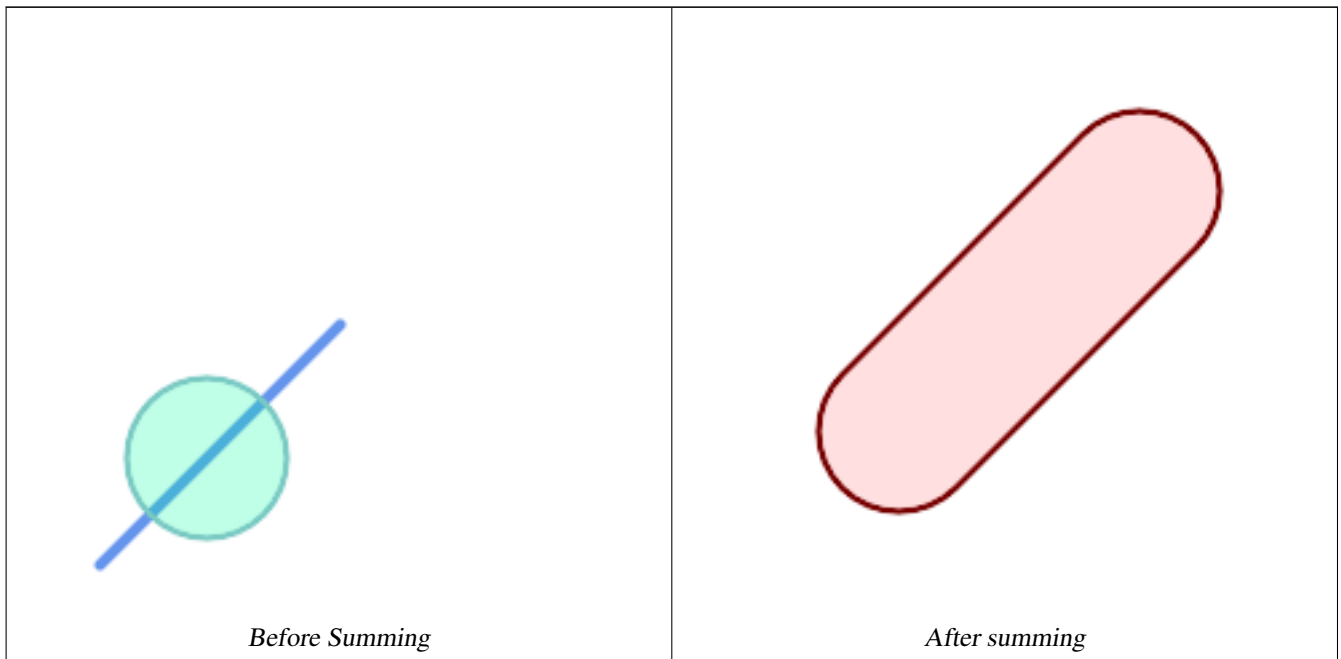


This method needs SFCGAL backend.

---

## Ejemplos

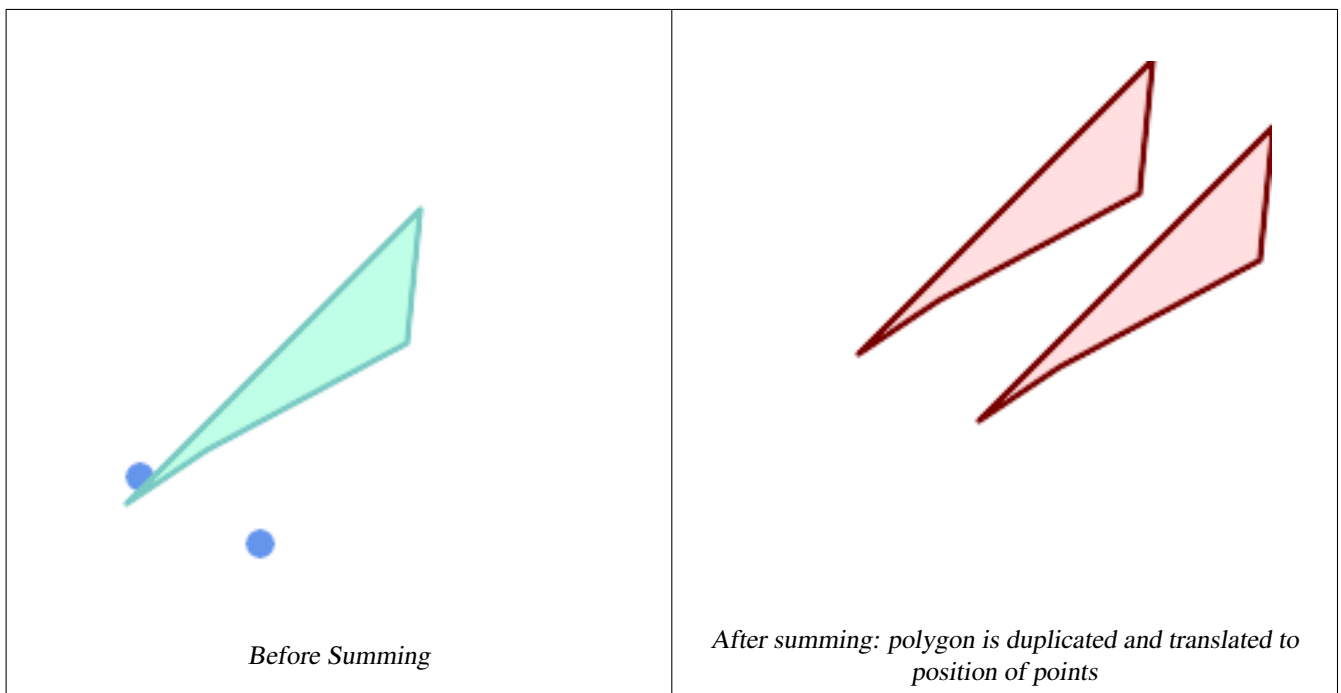
Minkowski Sum of Linestring and circle polygon where Linestring cuts thru the circle



```
SELECT ST_MinkowskiSum(line, circle))
FROM (SELECT
 ST_MakeLine(ST_Point(10, 10),ST_Point(100, 100)) As line,
 ST_Buffer(ST_GeomFromText('POINT(50 50)'), 30) As circle) As foo;

-- wkt --
MULTIPOLYGON(((30 59.9999999999999,30.5764415879031 54.1472903395161,32.2836140246614 ↵
 48.5194970290472,35.0559116309237 43.3328930094119,38.7867965644036 ↵
 38.7867965644035,43.332893009412 35.0559116309236,48.5194970290474 ↵
 32.2836140246614,54.1472903395162 30.5764415879031,60.0000000000001 30,65.8527096604839 ↵
 30.5764415879031,71.4805029709527 32.2836140246614,76.6671069905881 ↵
 35.0559116309237,81.2132034355964 38.7867965644036,171.213203435596 ↵
 128.786796564404,174.944088369076 133.332893009412,177.716385975339 ↵
 138.519497029047,179.423558412097 144.147290339516,180 150,179.423558412097 ↵
 155.852709660484,177.716385975339 161.480502970953,174.944088369076 ↵
 166.667106990588,171.213203435596 171.213203435596,166.667106990588 174.944088369076,
 161.480502970953 177.716385975339,155.852709660484 179.423558412097,150 ↵
 180,144.147290339516 179.423558412097,138.519497029047 177.716385975339,133.332893009412 ↵
 174.944088369076,128.786796564403 171.213203435596,38.7867965644035 ↵
 81.2132034355963,35.0559116309236 76.667106990588,32.2836140246614 ↵
 71.4805029709526,30.5764415879031 65.8527096604838,30 59.9999999999999)))
```

Minkowski Sum of a polygon and multipoint



```
SELECT ST_MinkowskiSum(mp, poly)
FROM (SELECT 'MULTIPOINT(25 50,70 25)::geometry As mp,
 'POLYGON((130 150, 20 40, 50 60, 125 100, 130 150))::geometry As poly
) As foo

-- wkt --
MULTIPOLYGON(
 ((70 115,100 135,175 175,225 225,70 115)),
 ((120 65,150 85,225 125,275 175,120 65))
)
```

### 7.21.17 ST\_OptimalAlphaShape

**ST\_OptimalAlphaShape** — Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.

#### Synopsis

geometry **ST\_OptimalAlphaShape**(geometry geom, boolean allow\_holes = false, integer nb\_components = 1);

#### Descripción

Computes the "optimal" alpha-shape of the points in a geometry. The alpha-shape is computed using a value of  $\alpha$  chosen so that:

1. the number of polygon elements is equal to or smaller than `nb_components` (which defaults to 1)
2. all input points are contained in the shape

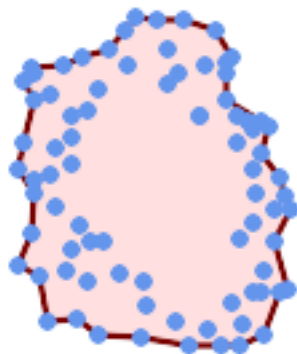
The result will not contain holes unless the optional `allow_holes` argument is specified as true.

Availability: 3.3.0 - requires SFCGAL >= 1.4.1.



This method needs SFCGAL backend.

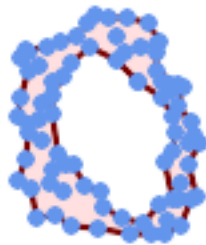
## Ejemplos



*Optimal alpha-shape of a MultiPoint (same example as [ST\\_AlphaShape](#))*

```
SELECT ST_AsText(ST_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ←
, (81 70),
 (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 ←
 30),(36 61),(32 65),
 (81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
 (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ←
 29),(27 84),(52 98),(72 95),(85 71),
 (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ←
 97),(27 77),(39 88),(60 81),
 (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ←
 64),(69 86),(60 90),(50 86),(43 80),(36 73),
 (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ←
 16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry));

POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ←
33,23 36,
 26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 ←
 97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53))
```



*Optimal alpha-shape of a MultiPoint, allowing holes (same example as [ST\\_AlphaShape](#))*

```
SELECT ST_AsText(ST_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ←
, (81 70),(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ←
, (36 61),(32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ←
29),(27 84),(52 98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ←
97),(27 77),(39 88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ←
64),(69 86),(60 90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ←
16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry, allow_holes =
> true));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ←
33,23 36,26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 ←
97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53),(36 61,36 68,40 75,43 ←
80,50 86,60 81,68 73,77 67,81 60,82 54,81 47,78 43,81 29,76 27,70 20,62 22,55 26,54 ←
32,48 34,44 42,38 46,36 61))
```

**Ver también**

[ST\\_ConcaveHull](#), [ST\\_AlphaShape](#)

## 7.21.18 ST\_Orientation

ST\_Orientation — Determine surface orientation

### Synopsis

integer **ST\_Orientation**(geometry geom);

### Descripción

The function only applies to polygons. It returns -1 if the polygon is counterclockwise oriented and 1 if the polygon is clockwise oriented.



Disponibilidad: 2.1.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.

### 7.21.19 ST\_StraightSkeleton

ST\_StraightSkeleton — Compute a straight skeleton from a geometry

#### Synopsis

geometry **ST\_StraightSkeleton**(geometry geom);

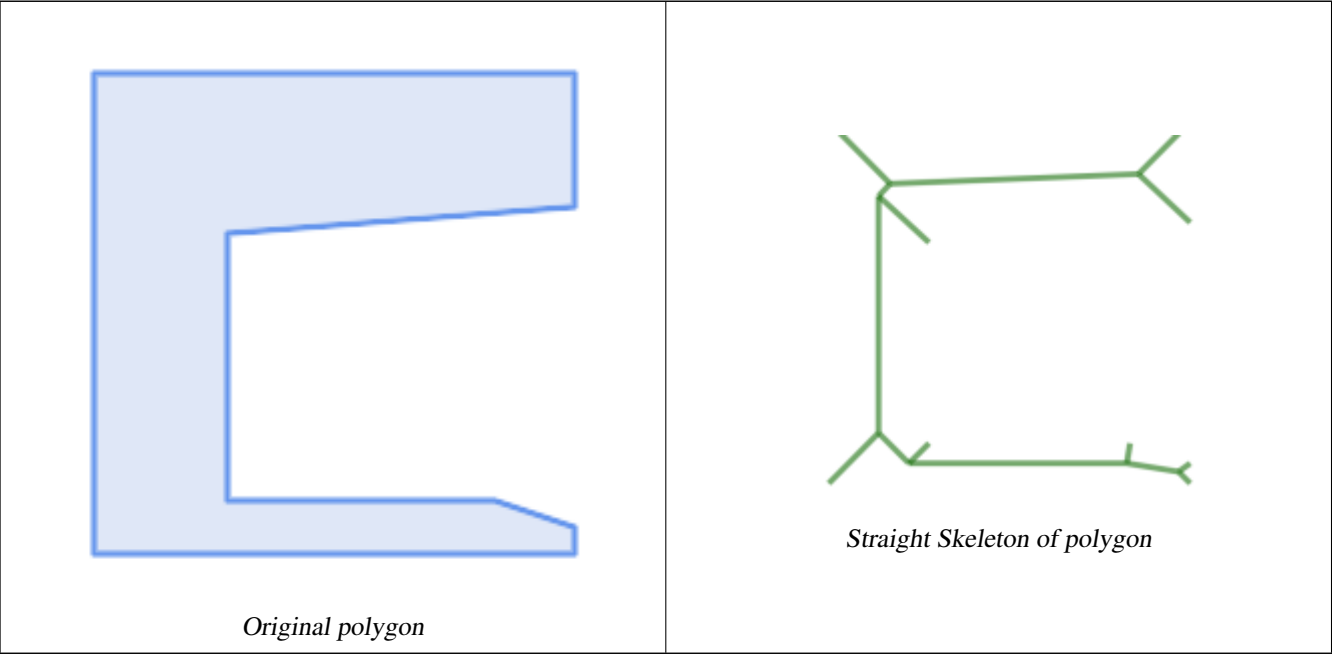
#### Descripción

Disponibilidad: 2.1.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

#### Ejemplos

```
SELECT ST_StraightSkeleton(ST_GeomFromText('POLYGON ((190 190, 10 190, 10 10, 190 10, 190 ←
20, 160 30, 60 30, 60 130, 190 140, 190 190))'));
```



### 7.21.20 ST\_Tessellate

ST\_Tessellate — Perform surface Tessellation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS

#### Synopsis

geometry **ST\_Tessellate**(geometry geom);

#### Descripción

Takes as input a surface such a MULTI(POLYGON) or POLYHEDRALSURFACE and returns a TIN representation via the process of tessellation using triangles.

Disponibilidad: 2.1.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.

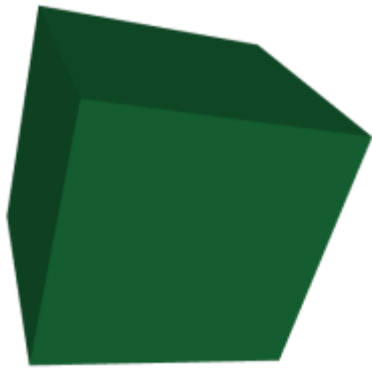


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

#### Ejemplos

---

```
SELECT ST_GeomFromText('POLYHEDRALSURFACE ↵
 Z(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)), ↵
 ((0 0 0, 0 1 0, 1 1 0, 1 ↵
0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 ↵
 ((1 1 0, 1 1 1, 1 0 1, 1 ↵
0 0, 1 1 0)), ↵
 ((0 1 0, 0 1 1, 1 1 1, 1 ↵
1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))))'))
```



Original Cube

```
SELECT ST_Tessellate(ST_GeomFromText(' ↵
 POLYHEDRALSURFACE Z(((0 0 0, 0 0 1, 0 1 1, 0 1 ↵
 ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 ↵
0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ↵
 ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 ↵
0)), ↵
 ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 ↵
0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))))'))
```

ST\_AsText output:

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



Tessellated Cube with triangles colored



- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 9.1 (same as ST\_3DVolume)

Ejemplo

When closed surfaces are created with WKT, they are treated as areal rather than solid. To make them solid, you need to use **ST\_MakeSolid**. Areal geometries have no volume. Here is an example to demonstrate.

```
SELECT ST_Volume(geom) As cube_surface_vol,
 ST_Volume(ST_MakeSolid(geom)) As solid_surface_vol
FROM (SELECT 'POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
 ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
 ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
 ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
 ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
 ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))::geometry' As f(geom);

cube_surface_vol | solid_surface_vol
-----+-----
0 | 1
```

Ver también

[ST\\_3DArea](#), [ST\\_MakeSolid](#), [ST\\_IsSolid](#)

## 7.22 Soporte para transacciones grandes



**Note**  
Los usuarios deben usar **el nivel de transacción en serie** sino se rompería el mecanismo de bloqueo.

### 7.22.1 AddAuth

AddAuth — Agrega un testigo de autorización para usarlo en la transacción actual.

Synopsis

boolean **AddAuth**(text auth\_token);

Descripción

Agrega un testigo de autorización para usarlo en la transacción actual.  
Adds the current transaction identifier and authorization token to a temporary table called temp\_lock\_have\_table.  
Disponibilidad: 1.1.3

## Ejemplos

```
SELECT LockRow('towns', '353', 'priscilla');
 BEGIN TRANSACTION;
 SELECT AddAuth('joey');
 UPDATE towns SET geom = ST_Translate(geom,2,2) WHERE gid = 353;
 COMMIT;

---Error--
ERROR: UPDATE where "gid" = '353' requires authorization 'priscilla'
```

## Ver también

[LockRow](#)

### 7.22.2 CheckAuth

**CheckAuth** — Crea un disparador sobre una tabla para prevenir/permitir actualizaciones y borrados de filas basados en el testigo de autorización.

## Synopsis

```
integer CheckAuth(text a_schema_name, text a_table_name, text a_key_column_name);
integer CheckAuth(text a_table_name, text a_key_column_name);
```

## Descripción

Crea un disparador sobre una tabla para prevenir/permitir actualizaciones y borrado de filas basado en el testigo de autorizaciones. Identifica filas usando la columna <rowid\_col> .

Si no se le pasa un nombre de esquema, a\_schema\_name, busca la tabla en el esquema actual.



### Note

Si ya existe un disparador de autorización sobre esta tabla la función da error.  
Si no está habilitado el soporte de transacciones, la función lanza una excepción.

Disponibilidad: 1.1.3

## Ejemplos

```
SELECT CheckAuth('public', 'towns', 'gid');
 result

 0
```

## Ver también

[EnableLongTransactions](#)

### 7.22.3 DisableLongTransactions

DisableLongTransactions — DisableLongTransactions

#### Synopsis

text **DisableLongTransactions**();

#### Descripción

Deshabilita el soporte de transacciones grandes. Esta función elimina las tablas de metadatos para soporte de transacciones grandes, y borra todos los disparadores vinculados a las tablas de comprobación de bloqueos.

Elimina la meta tabla llamada `authorization_table` y una vista llamada `authorized_tables` y todos los disparadores llamados `checkauthtrigger`

Disponibilidad: 1.1.3

#### Ejemplos

```
SELECT DisableLongTransactions();
--result--
Long transactions support disabled
```

#### Ver también

[EnableLongTransactions](#)

### 7.22.4 EnableLongTransactions

EnableLongTransactions — EnableLongTransactions

#### Synopsis

text **EnableLongTransactions**();

#### Descripción

Habilita el soporte de transacciones grandes. Esta función crea las tablas de metadatos requeridas, necesita ser llamada una vez antes de usar las otras funciones en esta sección. Llamarla más de una vez no produce problemas.

Crea una meta tabla llamada `authorization_table` y una vista llamada `authorized_tables`

Disponibilidad: 1.1.3

#### Ejemplos

```
SELECT EnableLongTransactions();
--result--
Long transactions support enabled
```

**Ver también**

[DisableLongTransactions](#)

### 7.22.5 LockRow

LockRow — Configura el bloqueo/autorización para una fila específica de la tabla

#### Synopsis

```
integer LockRow(text a_schema_name, text a_table_name, text a_row_key, text an_auth_token, timestamp expire_dt);
integer LockRow(text a_table_name, text a_row_key, text an_auth_token, timestamp expire_dt);
integer LockRow(text a_table_name, text a_row_key, text an_auth_token);
```

#### Descripción

Configura el bloqueo/autorización para una fila específica de la tabla <authid> es un valor de texto, <expires> es un valor de tiempo que por defecto es now()+1hora. Devuelve 1 si se asignó el bloqueo, 0 en otro caso (ya bloqueado por otra autorización)

Disponibilidad: 1.1.3

#### Ejemplos

```
SELECT LockRow('public', 'towns', '2', 'joey');
LockRow

1

--Joey has already locked the record and Priscilla is out of luck
SELECT LockRow('public', 'towns', '2', 'priscilla');
LockRow

0
```

**Ver también**

[UnlockRows](#)

### 7.22.6 UnlockRows

UnlockRows — Removes all locks held by an authorization token.

#### Synopsis

```
integer UnlockRows(text auth_token);
```

#### Descripción

Retira todos los bloqueos mantenidos por el id de la autorización especificada. Devuelve el número de bloqueos liberados.

Disponibilidad: 1.1.3

---



## Ejemplos

```
SELECT LockRow('towns', '353', 'priscilla');
 SELECT LockRow('towns', '2', 'priscilla');
 SELECT UnLockRows('priscilla');
 UnLockRows

 2
```

## Ver también

[LockRow](#)

## 7.23 Version Functions

### 7.23.1 PostGIS\_Extensions\_Upgrade

**PostGIS\_Extensions\_Upgrade** — Packages and upgrades PostGIS extensions (e.g. `postgis_raster`, `postgis_topology`, `postgis_sfcgal`) to given or latest version.

#### Synopsis

text **PostGIS\_Extensions\_Upgrade**(text target\_version=null);

#### Description

Packages and upgrades PostGIS extensions to given or latest version. Only extensions you have installed in the database will be packaged and upgraded if needed. Reports full PostGIS version and build configuration infos after. This is short-hand for doing multiple `CREATE EXTENSION .. FROM unpackaged` and `ALTER EXTENSION .. UPDATE` for each PostGIS extension. Currently only tries to upgrade extensions `postgis`, `postgis_raster`, `postgis_sfcgal`, `postgis_topology`, and `postgis_tiger_geocoder`.

Availability: 2.5.0



#### Note

Changed: 3.4.0 to add target\_version argument.

Changed: 3.3.0 support for upgrades from any PostGIS version. Does not work on all systems.

Changed: 3.0.0 to repackage loose extensions and support `postgis_raster`.

## Examples

```
SELECT PostGIS_Extensions_Upgrade();
```

```
NOTICE: Packaging extension postgis
NOTICE: Packaging extension postgis_raster
NOTICE: Packaging extension postgis_sfcgal
NOTICE: Extension postgis_topology is not available or not packagable for some reason
NOTICE: Extension postgis_tiger_geocoder is not available or not packagable for some reason
 reason

 postgis_extensions_upgrade

Upgrade completed, run SELECT postgis_full_version(); for details
(1 row)
```

**See Also**

Section 3.4, [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

**7.23.2 PostGIS\_Full\_Version**

`PostGIS_Full_Version` — Reports full PostGIS version and build configuration infos.

**Synopsis**

```
text PostGIS_Full_Version();
```

**Description**

Reports full PostGIS version and build configuration infos. Also informs about synchronization between libraries and scripts suggesting upgrades as needed.

Enhanced: 3.4.0 now includes extra PROJ configurations `NETWORK_ENABLED`, `URL_ENDPOINT` and `DATABASE_PATH` of `proj.db` location

**Examples**

```
SELECT PostGIS_Full_Version();
```

|                                                                                                                                                                                                                                                                                                                                                                                                                     | postgis_full_version |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|
| POSTGIS="3.4.0dev 3.3.0rc2-993-g61bdf43a7" [EXTENSION] PGSQL="160" GEOS="3.12.0dev-CAPI ↵<br>-1.18.0" SFCGAL="1.3.8" PROJ="7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj. ↵<br>org USER_WRITABLE_DIRECTORY=/tmp/proj DATABASE_PATH=/usr/share/proj/proj.db" GDAL="GDAL ↵<br>3.2.2, released 2021/03/05" LIBXML="2.9.10" LIBJSON="0.15" LIBPROTOBUF="1.3.3" WAGYU ↵<br>="0.5.0 (Internal)" TOPOLOGY RASTER |                      |

```
(1 row)
```

**See Also**

Section 3.4, [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Wagyu\\_Version](#), [PostGIS\\_Version](#)

**7.23.3 PostGIS\_GEOS\_Version**

`PostGIS_GEOS_Version` — Returns the version number of the GEOS library.

**Synopsis**

```
text PostGIS_GEOS_Version();
```

**Description**

Returns the version number of the GEOS library, or `NULL` if GEOS support is not enabled.

## Examples

```
SELECT PostGIS_GEOS_Version();
 postgis_geos_version

3.12.0dev-CAPI-1.18.0
(1 row)
```

## See Also

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

### 7.23.4 PostGIS\_GEOS\_Compiled\_Version

`PostGIS_GEOS_Compiled_Version` — Returns the version number of the GEOS library against which PostGIS was built.

## Synopsis

text `PostGIS_GEOS_Compiled_Version()`;

## Description

Returns the version number of the GEOS library, or against which PostGIS was built.

Availability: 3.4.0

## Examples

```
SELECT PostGIS_GEOS_Compiled_Version();
 postgis_geos_compiled_version

3.12.0
(1 row)
```

## See Also

[PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Full\\_Version](#)

### 7.23.5 PostGIS\_Liblwgeom\_Version

`PostGIS_Liblwgeom_Version` — Returns the version number of the liblwgeom library. This should match the version of PostGIS.

## Synopsis

text `PostGIS_Liblwgeom_Version()`;

## Description

Returns the version number of the liblwgeom library/

---

## Examples

```
SELECT PostGIS_Liblwgeom_Version();
postgis_liblwgeom_version

3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

## See Also

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

### 7.23.6 PostGIS\_LibXML\_Version

`PostGIS_LibXML_Version` — Returns the version number of the libxml2 library.

## Synopsis

text `PostGIS_LibXML_Version()`;

## Description

Returns the version number of the LibXML2 library.

Availability: 1.5

## Examples

```
SELECT PostGIS_LibXML_Version();
postgis_libxml_version

2.9.10
(1 row)
```

## See Also

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Version](#)

### 7.23.7 PostGIS\_Lib\_Build\_Date

`PostGIS_Lib_Build_Date` — Returns build date of the PostGIS library.

## Synopsis

text `PostGIS_Lib_Build_Date()`;

## Description

Returns build date of the PostGIS library.

---

## Examples

```
SELECT PostGIS_Lib_Build_Date();
 postgis_lib_build_date

2023-06-22 03:56:11
(1 row)
```

### 7.23.8 PostGIS\_Lib\_Version

PostGIS\_Lib\_Version — Returns the version number of the PostGIS library.

#### Synopsis

text **PostGIS\_Lib\_Version()**;

#### Description

Returns the version number of the PostGIS library.

#### Examples

```
SELECT PostGIS_Lib_Version();
 postgis_lib_version

3.4.0dev
(1 row)
```

#### See Also

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

### 7.23.9 PostGIS\_PROJ\_Version

PostGIS\_PROJ\_Version — Returns the version number of the PROJ4 library.

#### Synopsis

text **PostGIS\_PROJ\_Version()**;

#### Description

Returns the version number of the PROJ library and some configuration options of proj.

Enhanced: 3.4.0 now includes NETWORK\_ENABLED, URL\_ENDPOINT and DATABASE\_PATH of proj.db location

## Examples

```
SELECT PostGIS_PROJ_Version();
 postgis_proj_version

7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org USER_WRITABLE_DIRECTORY=/tmp/ ↵
 proj DATABASE_PATH=/usr/share/proj/proj.db
(1 row)
```

## See Also

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_Version](#)

### 7.23.10 PostGIS\_Wagyu\_Version

`PostGIS_Wagyu_Version` — Returns the version number of the internal Wagyu library.

## Synopsis

text `PostGIS_Wagyu_Version()`;

## Description

Returns the version number of the internal Wagyu library, or NULL if Wagyu support is not enabled.

## Examples

```
SELECT PostGIS_Wagyu_Version();
 postgis_wagyu_version

0.5.0 (Internal)
(1 row)
```

## See Also

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_Version](#)

### 7.23.11 PostGIS\_Scripts\_Build\_Date

`PostGIS_Scripts_Build_Date` — Returns build date of the PostGIS scripts.

## Synopsis

text `PostGIS_Scripts_Build_Date()`;

## Description

Returns build date of the PostGIS scripts.

Availability: 1.0.0RC1

---

## Examples

```
SELECT PostGIS_Scripts_Build_Date();
 postgis_scripts_build_date

2023-06-22 03:56:11
(1 row)
```

## See Also

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_Version](#)

### 7.23.12 PostGIS\_Scripts\_Installed

`PostGIS_Scripts_Installed` — Returns version of the PostGIS scripts installed in this database.

## Synopsis

text `PostGIS_Scripts_Installed()`;

## Description

Returns version of the PostGIS scripts installed in this database.



### Note

If the output of this function doesn't match the output of [PostGIS\\_Scripts\\_Released](#) you probably missed to properly upgrade an existing database. See the [Upgrading](#) section for more info.

Availability: 0.9.0

## Examples

```
SELECT PostGIS_Scripts_Installed();
 postgis_scripts_installed

3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

## See Also

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Scripts\\_Released](#), [PostGIS\\_Version](#)

### 7.23.13 PostGIS\_Scripts\_Released

`PostGIS_Scripts_Released` — Returns the version number of the `postgis.sql` script released with the installed PostGIS lib.

## Synopsis

text `PostGIS_Scripts_Released()`;

**Description**

Returns the version number of the postgis.sql script released with the installed PostGIS lib.

**Note**

Starting with version 1.1.0 this function returns the same value of `PostGIS_Lib_Version`. Kept for backward compatibility.

Availability: 0.9.0

**Examples**

```
SELECT PostGIS_Scripts_Released();
 postgis_scripts_released

3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

**See Also**

`PostGIS_Full_Version`, `PostGIS_Scripts_Installed`, `PostGIS_Lib_Version`

**7.23.14 PostGIS\_Version**

`PostGIS_Version` — Returns PostGIS version number and compile-time options.

**Synopsis**

text `PostGIS_Version()`;

**Description**

Returns PostGIS version number and compile-time options.

**Examples**

```
SELECT PostGIS_Version();
 postgis_version

3.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
(1 row)
```

**See Also**

`PostGIS_Full_Version`, `PostGIS_GEOS_Version`, `PostGIS_Lib_Version`, `PostGIS_LibXML_Version`, `PostGIS_PROJ_Version`



## 7.24 Grand Unified Custom Variables (GUCs)

### 7.24.1 postgis.backend

`postgis.backend` — The backend to service a function where GEOS and SFCGAL overlap. Options: `geos` or `sfcgal`. Defaults to `geos`.

#### Descripción

This GUC is only relevant if you compiled PostGIS with `sfcgal` support. By default `geos` backend is used for functions where both GEOS and SFCGAL have the same named function. This variable allows you to override and make `sfcgal` the backend to service the request.

Disponibilidad: 2.1.0

#### Ejemplos

Sets backend just for life of connection

```
set postgis.backend = sfcgal;
```

Sets backend for new connections to database

```
ALTER DATABASE mygisdb SET postgis.backend = sfcgal;
```

#### Ver también

Section [7.21](#)

### 7.24.2 postgis.gdal\_datapath

`postgis.gdal_datapath` — A configuration option to assign the value of GDAL's `GDAL_DATA` option. If not set, the environmentally set `GDAL_DATA` variable is used.

#### Descripción

A PostgreSQL GUC variable for setting the value of GDAL's `GDAL_DATA` option. The `postgis.gdal_datapath` value should be the complete physical path to GDAL's data files.

This configuration option is of most use for Windows platforms where GDAL's data files path is not hard-coded. This option should also be set when GDAL's data files are not located in GDAL's expected path.



#### Note

This option can be set in PostgreSQL's configuration file `postgresql.conf`. It can also be set by connection or transaction.

Disponibilidad: 2.2.0



#### Note

Additional information about `GDAL_DATA` is available at GDAL's [Configuration Options](#).

---

## Ejemplos

Set and reset `postgis.gdal_datapath`

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';
SET postgis.gdal_datapath TO default;
```

Setting on windows for a particular database

```
ALTER DATABASE gisdb
SET postgis.gdal_datapath = 'C:/Program Files/PostgreSQL/9.3/gdal-data';
```

## Ver también

[PostGIS\\_GDAL\\_Version](#), [ST\\_Transform](#)

### 7.24.3 `postgis.gdal_enabled_drivers`

`postgis.gdal_enabled_drivers` — A configuration option to set the enabled GDAL drivers in the PostGIS environment. Affects the GDAL configuration variable `GDAL_SKIP`.

#### Descripción

A configuration option to set the enabled GDAL drivers in the PostGIS environment. Affects the GDAL configuration variable `GDAL_SKIP`. This option can be set in PostgreSQL's configuration file: `postgresql.conf`. It can also be set by connection or transaction.

The initial value of `postgis.gdal_enabled_drivers` may also be set by passing the environment variable `POSTGIS_GDAL_ENABLED_DRIVERS` with the list of enabled drivers to the process starting PostgreSQL.

Enabled GDAL specified drivers can be specified by the driver's short-name or code. Driver short-names or codes can be found at [GDAL Raster Formats](#). Multiple drivers can be specified by putting a space between each driver.

#### Note

There are three special codes available for `postgis.gdal_enabled_drivers`. The codes are case-sensitive.

- `DISABLE_ALL` disables all GDAL drivers. If present, `DISABLE_ALL` overrides all other values in `postgis.gdal_enabled_drivers`.
- `ENABLE_ALL` enables all GDAL drivers.
- `VSICURL` enables GDAL's `/vsicurl/` virtual file system.

When `postgis.gdal_enabled_drivers` is set to `DISABLE_ALL`, attempts to use out-db rasters, `ST_FromGDALRaster()`, `ST_AsGDALRaster()`, `ST_AsTIFF()`, `ST_AsJPEG()` and `ST_AsPNG()` will result in error messages.



#### Note

In the standard PostGIS installation, `postgis.gdal_enabled_drivers` is set to `DISABLE_ALL`.



#### Note

Additional information about `GDAL_SKIP` is available at GDAL's [Configuration Options](#).

Disponibilidad: 2.2.0

## Ejemplos

Set and reset `postgis.gdal_enabled_drivers`

Sets backend for all new connections to database

```
ALTER DATABASE mygisdb SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

Sets default enabled drivers for all new connections to server. Requires super user access and PostgreSQL 9.4+. Also note that database, session, and user settings override this.

```
ALTER SYSTEM SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
SELECT pg_reload_conf();
```

```
SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
SET postgis.gdal_enabled_drivers = default;
```

Enable all GDAL Drivers

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
```

Disable all GDAL Drivers

```
SET postgis.gdal_enabled_drivers = 'DISABLE_ALL';
```

## Ver también

[ST\\_FromGDALRaster](#), [ST\\_AsGDALRaster](#), [ST\\_AsTIFF](#), [ST\\_AsPNG](#), [ST\\_AsJPEG](#), [postgis.enable\\_outdb\\_rasters](#)

### 7.24.4 `postgis.enable_outdb_rasters`

`postgis.enable_outdb_rasters` — A boolean configuration option to enable access to out-db raster bands.

#### Descripción

A boolean configuration option to enable access to out-db raster bands. This option can be set in PostgreSQL's configuration file: `postgresql.conf`. It can also be set by connection or transaction.

The initial value of `postgis.enable_outdb_rasters` may also be set by passing the environment variable `POSTGIS_ENABLE_OUTDB_RASTERS` with a non-zero value to the process starting PostgreSQL.



#### Note

Even if `postgis.enable_outdb_rasters` is `True`, the GUC `postgis.gdal_enabled_drivers` determines the accessible raster formats.



#### Note

In the standard PostGIS installation, `postgis.enable_outdb_rasters` is set to `False`.

Disponibilidad: 2.2.0

## Ejemplos

Set and reset `postgis.enable_outdb_rasters` for current session

```
SET postgis.enable_outdb_rasters TO True;
SET postgis.enable_outdb_rasters = default;
SET postgis.enable_outdb_rasters = True;
SET postgis.enable_outdb_rasters = False;
```

Set for specific database

```
ALTER DATABASE gisdb SET postgis.enable_outdb_rasters = true;
```

Setting for whole database cluster. You need to reconnect to the database for changes to take effect.

```
--writes to postgres.auto.conf
ALTER SYSTEM postgis.enable_outdb_rasters = true;
--Reloads postgres conf
SELECT pg_reload_conf();
```

## Ver también

[postgis.gdal\\_enabled\\_drivers](#) [postgis.gdal\\_config\\_options](#)

## 7.24.5 postgis.gdal\_config\_options

`postgis.gdal_config_options` — A string configuration to set options used when working with an out-db raster.

## Descripción

A string configuration to set options used when working with an out-db raster. [Configuration options](#) control things like how much space GDAL allocates to local data cache, whether to read overviews, and what access keys to use for remote out-db data sources.

Disponibilidad: 2.2.0

## Ejemplos

Set `postgis.gdal_vsi_options` for current session:

```
SET postgis.gdal_config_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY= ↵
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYY';
```

Set `postgis.gdal_vsi_options` just for the *current transaction* using the `LOCAL` keyword:

```
SET LOCAL postgis.gdal_config_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxx ↵
AWS_SECRET_ACCESS_KEY=YYYYYYYYYYYYYYYYYYYYYYYYYYYYYY';
```

## Ver también

[postgis.enable\\_outdb\\_rasters](#) [postgis.gdal\\_enabled\\_drivers](#)

## 7.25 Troubleshooting Functions

### 7.25.1 PostGIS\_AddBBox

PostGIS\_AddBBox — Add bounding box to the geometry.

#### Synopsis

geometry **PostGIS\_AddBBox**(geometry geomA);

#### Description

Add bounding box to the geometry. This would make bounding box based queries faster, but will increase the size of the geometry.



#### Note

Bounding boxes are automatically added to geometries so in general this is not needed unless the generated bounding box somehow becomes corrupted or you have an old install that is lacking bounding boxes. Then you need to drop the old and readd.



This method supports Circular Strings and Curves.

#### Examples

```
UPDATE sometable
SET geom = PostGIS_AddBBox(geom)
WHERE PostGIS_HasBBox(geom) = false;
```

#### See Also

[PostGIS\\_DropBBox](#), [PostGIS\\_HasBBox](#)

### 7.25.2 PostGIS\_DropBBox

PostGIS\_DropBBox — Drop the bounding box cache from the geometry.

#### Synopsis

geometry **PostGIS\_DropBBox**(geometry geomA);

#### Description

Drop the bounding box cache from the geometry. This reduces geometry size, but makes bounding-box based queries slower. It is also used to drop a corrupt bounding box. A tale-tell sign of a corrupt cached bounding box is when your ST\_Intersects and other relation queries leave out geometries that rightfully should return true.

**Note**

Bounding boxes are automatically added to geometries and improve speed of queries so in general this is not needed unless the generated bounding box somehow becomes corrupted or you have an old install that is lacking bounding boxes. Then you need to drop the old and readd. This kind of corruption has been observed in 8.3-8.3.6 series whereby cached bboxes were not always recalculated when a geometry changed and upgrading to a newer version without a dump reload will not correct already corrupted boxes. So one can manually correct using below and readd the bbox or do a dump reload.



This method supports Circular Strings and Curves.

**Examples**

```
--This example drops bounding boxes where the cached box is not correct
--The force to ST_AsBinary before applying Box2D forces a ↔
 recalculation of the box, and Box2D applied to the table ↔
 geometry always
-- returns the cached bounding box.
UPDATE sometable
SET geom = PostGIS_DropBBox(geom)
WHERE Not (Box2D(ST_AsBinary(geom)) = Box2D(geom));

UPDATE sometable
SET geom = PostGIS_AddBBox(geom)
WHERE Not PostGIS_HasBBOX(geom);
```

**See Also**

[PostGIS\\_AddBBox](#), [PostGIS\\_HasBBox](#), [Box2D](#)

**7.25.3 PostGIS\_HasBBox**

**PostGIS\_HasBBox** — Returns TRUE if the bbox of this geometry is cached, FALSE otherwise.

**Synopsis**

boolean **PostGIS\_HasBBox**(geometry geomA);

**Description**

Returns TRUE if the bbox of this geometry is cached, FALSE otherwise. Use [PostGIS\\_AddBBox](#) and [PostGIS\\_DropBBox](#) to control caching.



This method supports Circular Strings and Curves.

**Examples**

```
SELECT geom
FROM sometable WHERE PostGIS_HasBBox(geom) = false;
```

**See Also**

[PostGIS\\_AddBBox](#), [PostGIS\\_DropBBox](#)

## Chapter 8

# Topology

Los tipos y funciones de PostGIS Topology son usados para manejar objetos topológicos tales como caras, bordes y nodos.

La presentación de Sandro Santilli en la conferencia PostGIS Day Paris 2011 da una buena sinopsis de la Topología PostGIS y hacia donde se dirige [Topology with PostGIS 2.0 slide deck](#).

Vincent Picavet provides a good synopsis and overview of what is Topology, how is it used, and various FOSS4G tools that support it in [PostGIS Topology PGConf EU 2012](#).

Un ejemplo de una base de datos SIG basado topologicamente en la base de datos del [Sistema de Codificación y Referencia Geográfica Topologicamente Integrado del Censo de US \(TIGER\)](#) . Si desea experimentar con la topología de PostGIS y nesecita algunos datos, ver [Topology\\_Load\\_Tiger](#).

El módulo de topología ha existido en versiones anteriores de PostGIS pero nunca hizo parte de la documentación oficial. En PostGIS 2.0.0 una limpieza a gran escala está teniendo lugar con el fin de eliminar el uso de todas las funciones obsoletas, solucionar los problemas de usabilidad conocidos, documentar mejor las características y funciones, agregar nuevas funciones y mejorarlo para satisfacer más de cerca los estándares SQL-MM.

Detalles de este proyecto pueden encontrarse en [PostGIS Topology Wiki](#)

Todas las funciones y tablas asociadas con este módulo son instaladas en un esquema llamado `topology`

Las funciones que son definidas bajo el estandar SQL/MM son prefijadas con `ST_` y las funciones específicas a PostGIS no son prefijadas.

El soporte de topología se crea de manera predeterminada a partir de PostGIS 2.0, y se puede deshabilitar especificando la opción de configuración `--without-topology` en tiempo de compilación como se describe en [Chapter 2](#)

## 8.1 Tipos en Topology

### 8.1.1 `getfaceedges_returntype`

`getfaceedges_returntype` — Un tipo compuesto que necesita un número de secuencia y un número de eje.

#### Descripción

Un tipo compuesto que consiste de un número de secuencia y de un número de eje. Es el tipo devuelto por las funciones `ST_GetFaceEdges` y `GetNodeEdges` .

1. `sequence` es un entero: Se refiere a una topología definida en la tabla `topology.topology` la cual define el esquema y el srid de la topología.
  2. `edge` es un entero. El identificador de un borde.
-



### 8.1.2 TopoGeometry

TopoGeometry — Un tipo compuesto que representa una geometría topológicamente definida.

#### Descripción

Un tipo compuesto que se refiere a una geometría de topología en una capa específica de la topología, con un tipo y un id específicos. Los elementos de una TopoGeometry son las propiedades: topology\_id, layer\_id, id integer, type integer.

1. `topology_id` es un entero: Se refiere a una topología definida en la tabla `topology.topology`, la cual define el esquema y el srid de la topología.
2. `layer_id` es un entero: El `layer_id` en la tabla `layers` a la que pertenece TopoGeometry. La combinación de `topology_id` y `layer_id` provee una referencia única en la tabla `topology.layers`.
3. `id` es un entero: El id es el número autogenerated secuencialmente que define de manera única la topogeometría en la respectiva capa de topología.
4. `type` entero entre 1 y 4 que define el tipo de geometría: 1:[multi]punto, 2:[multi]línea, 3:[multi]polinomio, 4:colección.

#### Comportamiento de la conversión de tipos de dato

Esta sección lista las conversiones tanto automáticas como explícitas para este tipo de datos.

|             |                |
|-------------|----------------|
| Convertir a | Comportamiento |
| geometría   | automática     |

#### Ver también

[CreateTopoGeom](#)

### 8.1.3 validate\_topology\_returntype

`validate_topology_returntype` — Un tipo compuesto que consta de un mensaje de error e `id1` e `id2` para denotar la ubicación del error. Este es el tipo de valor devuelto por `ValidateTopology`.

#### Descripción

Un tipo compuesto que consiste de un mensaje de error y dos enteros. La función [ValidateTopology](#) devuelve un conjunto de estos para indicar errores en la validación y el `id1` e `id2` para indicar los identificadores de los objetos topológicos involucrados en el error.

1. `error` es varchar: Indica el tipo de error.  
Los descriptores de error actuales son: nodos coincidentes, edge crosses node, borde no simple, edge end node geometry mis-match, edge start node geometry mismatch, cara solapada, cara dentro de cara,
2. `id1` es un integer: Indica el identificador del borde / cara / nodo en error.
3. `id2` es un integer: Para errores que involucren 2 objetos indica el borde o nodo secundario.

#### Ver también

[ValidateTopology](#)

## 8.2 Dominios de Topology

### 8.2.1 TopoElement

TopoElement — Una matriz de 2 enteros usada generalmente para identificar un componente TopoGeometry

#### Descripción

Una matriz de 2 enteros usada para representar un componente de una **TopoGeometry** simple o jerárquica.

En el caso de una simple TopoGeometry el primer elemento de la matriz representa el identificador de una primitiva topológica y el segundo elemento representa su tipo (1: nodo, 2: borde, 3: cara). En el caso de una TopoGeometry jerárquica el primer elemento de la matriz representa el identificador de una TopoGeometry hijo y el segundo elemento representa su identificador de capa.



#### Note

Para cualquier TopoGeometry jerarquica dada todos los elementos TopoGeometry secundarios vendrán de la misma capa secundaria, tal como se especifica en el registro topology.layer para la capa de la TopoGeometry que se está definiendo.

#### Ejemplos

```
SELECT te[1] AS id, te[2] AS type FROM
(SELECT ARRAY[1,2]::topology.topoelement AS te) f;
 id | type
----+-----
 1 | 2
```

```
SELECT ARRAY[1,2]::topology.topoelement;
 te

{1,2}
```

```
--Example of what happens when you try to case a 3 element array to topoelement
-- NOTE: topoement has to be a 2 element array so fails dimension check
SELECT ARRAY[1,2,3]::topology.topoelement;
ERROR: value for domain topology.topoelement violates check constraint "dimensions"
```

#### Ver también

[GetTopoGeomElements](#), [TopoElementArray](#), [TopoGeometry](#), [TopoGeom\\_addElement](#), [TopoGeom\\_remElement](#)

### 8.2.2 TopoElementArray

TopoElementArray — Matriz de objetos TopoElement.

#### Descripción

Una matriz de 1 o más objetos TopoElement, generalmente usada para pasar al rededor de componentes de objetos TopoGeometry.

## Ejemplos

```
SELECT '{{1,2},{4,3}}'::topology.topoelementarray As tea;
 tea

{{1,2},{4,3}}

-- more verbose equivalent --
SELECT ARRAY[ARRAY[1,2], ARRAY[4,3]]::topology.topoelementarray As tea;

 tea

{{1,2},{4,3}}

--using the array agg function packaged with topology --
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
 FROM generate_series(1,4) As e CROSS JOIN generate_series(1,3) As t;
 tea

{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3},{4,1},{4,2},{4,3}}
```

```
SELECT '{{1,2,4},{3,4,5}}'::topology.topoelementarray As tea;
ERROR: value for domain topology.topoelementarray violates check constraint "dimensions"
```

## Ver también

[TopoElement](#), [GetTopoGeomElementArray](#), [TopoElementArray\\_Agg](#)

## 8.3 Topología y Gestión de TopoGeometría

### 8.3.1 AddTopoGeometryColumn

**AddTopoGeometryColumn** — Agrega una columna topogeometry a una tabla existente, registra esta nueva columna como una capa en topology.layer y devuelve el nuevo layer\_id

#### Synopsis

```
integer AddTopoGeometryColumn(varchar topology_name, varchar schema_name, varchar table_name, varchar column_name,
varchar feature_type);
integer AddTopoGeometryColumn(varchar topology_name, varchar schema_name, varchar table_name, varchar column_name,
varchar feature_type, integer child_layer);
```

#### Descripción

Cada objeto TopoGeometry pertenece a una capa específica de una topología específica. Antes de crear un objeto TopoGeometry usted necesita crear un TopologyLayer. Una capa de topología es una asociación de una tabla de características con la topología. También contiene información de tipo y jerarquía. Se crea una capa usando la función AddTopoGeometryColumn()

Esta función agregará la columna solicitada a la tabla y agregará un registro a la tabla topology.layer con toda la información dada.

Si no especifica [child\_layer] (o lo establece en NULL), esta capa contendrá TopoGeometría Básicas (compuesta por elementos de topología primitiva). De lo contrario, esta capa contendrá TopoGeometrias jerárquicas (compuestas por TopoGeometrias de la child\_layer).

Una vez creada la capa (su id es devuelto por la función `AddTopoGeometryColumn`), ya está listo para construir objetos TopoGeometry

Valid `feature_types` are: POINT, MULTIPOINT, LINE, MULTILINE, POLYGON, MULTIPOLYGON, COLLECTION

Availability: 1.1

### Ejemplos

```
-- Note for this example we created our new table in the ma_topo schema
-- though we could have created it in a different schema -- in which case topology_name and ←
-- schema_name would be different
CREATE SCHEMA ma;
CREATE TABLE ma.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('ma_topo', 'ma', 'parcels', 'topo', 'POLYGON');

CREATE SCHEMA ri;
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);
SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

### Ver también

[DropTopoGeometryColumn](#), [toTopoGeom](#), [CreateTopology](#), [CreateTopoGeom](#)

## 8.3.2 RenameTopoGeometryColumn

`RenameTopoGeometryColumn` — Renames a topogeometry column

### Synopsis

`topology.layer` **`RenameTopoGeometryColumn`**(regclass layer\_table, name feature\_column, name new\_name);

### Descripción

This function changes the name of an existing TopoGeometry column ensuring metadata information about it is updated accordingly.

Availability: 3.4.0

### Ejemplos

```
SELECT topology.RenameTopoGeometryColumn('public.parcels', 'topogeom', 'tgeom');
```

### Ver también

[AddTopoGeometryColumn](#), [RenameTopology](#)

## 8.3.3 DropTopology

`DropTopology` — Usar con precaución: Permite eliminar un esquema de topología y elimina su referencia de la tabla `topology.topology` y referencias a las tablas en ese esquema desde la tabla `geometry_columns`

## Synopsis

integer **DropTopology**(varchar topology\_schema\_name);

## Descripción

Permite eliminar un esquema de topología y elimina esta referencia de la tabla topology.topology y referencias a tablas en este esquema desde la tabla geometry\_columns. Esta función se debe USAR CON PRECAUCION, ya que podría destruir los datos que le interesan. Si el esquema no existe, simplemente elimina las entradas de referencia del esquema nombrado.

Availability: 1.1

## Ejemplos

Elimina en cascada el esquema ma\_topo y remueve todas las referencias a él en topology.topology y geometry\_columns.

```
SELECT topology.DropTopology('ma_topo');
```

## Ver también

[DropTopoGeometryColumn](#)

## 8.3.4 RenameTopology

RenameTopology — Renames a topology

## Synopsis

varchar **RenameTopology**(varchar old\_name, varchar new\_name);

## Descripción

Renames a topology schema, updating its metadata record in the topology.topology table.

Availability: 3.4.0

## Ejemplos

Rename a topology from topo\_stage to topo\_prod.

```
SELECT topology.RenameTopology('topo_stage', 'topo_prod');
```

## Ver también

[CopyTopology](#), [RenameTopoGeometryColumn](#)

## 8.3.5 DropTopoGeometryColumn

DropTopoGeometryColumn — Elimina la columna topogeometry de la tabla nombrada en table\_name en el esquema schema\_name y anula el registro de las columnas de la tabla topology.layer.

## Synopsis

text **DropTopoGeometryColumn**(varchar schema\_name, varchar table\_name, varchar column\_name);

## Descripción

Borra la columna de topogeometría de la tabla denominada `table_name` en el esquema `schema_name` y anula el registro de las columnas de la tabla `topology.layer`. Devuelve el resumen del estado de la eliminación. **NOTA:** Primero establece todos los valores en NULL antes de eliminar para omitir comprobaciones de integridad referencial.

Availability: 1.1

## Ejemplos

```
SELECT topology.DropTopoGeometryColumn('ma_topo', 'parcel_topo', 'topo');
```

## Ver también

[AddTopoGeometryColumn](#)

## 8.3.6 Populate\_Topology\_Layer

`Populate_Topology_Layer` — Agrega entradas faltantes a la tabla `topology.layer` mediante la lectura de metadatos de las tablas de topo.

## Synopsis

setof record **Populate\_Topology\_Layer**();

## Descripción

Agrega las entradas faltantes a la tabla `topology.layer` inspeccionando las restricciones de topología en las tablas. Esta función es útil para arreglar las entradas de catálogo de topología después de la restauración de esquemas con datos topo.

Devuelve la lista de entradas creadas. Las columnas devueltas son `schema_name`, `table_name`, `feature_column`.

Disponibilidad: 2.3.0

## Ejemplos

```
SELECT CreateTopology('strk_topo');
CREATE SCHEMA strk;
CREATE TABLE strk.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('strk_topo', 'strk', 'parcels', 'topo', 'POLYGON');
-- this will return no records because this feature is already registered
SELECT *
 FROM topology.Populate_Topology_Layer();

-- let's rebuild
TRUNCATE TABLE topology.layer;

SELECT *
 FROM topology.Populate_Topology_Layer();
```

```
SELECT topology_id, layer_id, schema_name As sn, table_name As tn, feature_column As fc
FROM topology.layer;
```

```
schema_name | table_name | feature_column
-----+-----+-----
strk | parcels | topo
(1 row)

topology_id | layer_id | sn | tn | fc
-----+-----+---+---+---
 2 | 2 | strk | parcels | topo
(1 row)
```

#### Ver también

[AddTopoGeometryColumn](#)

### 8.3.7 TopologySummary

TopologySummary — Toma un nombre de topología y proporciona totales de resumen de tipos de objetos en la topología.

#### Synopsis

text **TopologySummary**(varchar topology\_schema\_name);

#### Descripción

Toma un nombre de topología y proporciona totales de resumen de tipos de objetos en la topología.

Disponibilidad: 2.0.0

#### Ejemplos

```
SELECT topology.topologysummary('city_data');
 topologysummary

Topology city_data (329), SRID 4326, precision: 0
22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
Layer 1, type Polygonal (3), 9 topogeoms
 Deploy: features.land_parcels.feature
Layer 2, type Puntal (1), 8 topogeoms
 Deploy: features.traffic_signs.feature
Layer 3, type Lineal (2), 8 topogeoms
 Deploy: features.city_streets.feature
Layer 4, type Polygonal (3), 3 topogeoms
 Hierarchy level 1, child layer 1
 Deploy: features.big_parcels.feature
Layer 5, type Puntal (1), 1 topogeoms
 Hierarchy level 1, child layer 2
 Deploy: features.big_signs.feature
```

#### Ver también

[Topology\\_Load\\_Tiger](#)

### 8.3.8 ValidateTopology

ValidateTopology — Devuelve un conjunto de objetos `validate_topology_returntype` que detallan problemas con la topología.

#### Synopsis

setof `validate_topology_returntype` **ValidateTopology**(varchar toponame, geometry bbox);

#### Descripción

Returns a set of `validate_topology_returntype` objects detailing issues with topology, optionally limiting the check to the area specified by the `bbox` parameter.

List of possible errors, what they mean and what the returned ids represent are displayed below:

| Error                                                  | id1                                                                                    | id2                                     | Meaning                                                                                                                                                      |
|--------------------------------------------------------|----------------------------------------------------------------------------------------|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| coincident nodes                                       | Identifier of first node.                                                              | Identifier of second node.              | Two nodes have the same geometry.                                                                                                                            |
| Borde curza nodo                                       | Identifier of the edge.                                                                | Identifier of the node.                 | An edge has a node in its interior. See <a href="#">ST_Relate</a> .                                                                                          |
| borde inválido                                         | Identifier of the edge.                                                                |                                         | An edge geometry is invalid. See <a href="#">ST_IsValid</a> .                                                                                                |
| borde no simple                                        | Identifier of the edge.                                                                |                                         | An edge geometry has self-intersections. See <a href="#">ST_IsSimple</a> .                                                                                   |
| borde cruza borde                                      | Identifier of first edge.                                                              | Identifier of second edge.              | Two edges have an interior intersection. See <a href="#">ST_Relate</a> .                                                                                     |
| No coincide la geometría del nodo inicial de la arista | Identifier of the edge.                                                                | Identifier of the indicated start node. | The geometry of the node indicated as the starting node for an edge does not match the first point of the edge geometry. See <a href="#">ST_StartPoint</a> . |
| No coincide la geometría del nodo final de la arista   | Identifier of the edge.                                                                | Identifier of the indicated end node.   | The geometry of the node indicated as the ending node for an edge does not match the last point of the edge geometry. See <a href="#">ST_EndPoint</a> .      |
| Cara sin bordes                                        | Identifier of the orphaned face.                                                       |                                         | No edge reports an existing face on either of its sides ( <code>left_face</code> , <code>right_face</code> ).                                                |
| cara no tiene anillos                                  | Identifier of the partially-defined face.                                              |                                         | Edges reporting a face on their sides do not form a ring.                                                                                                    |
| face has wrong mbr                                     | Identifier of the face with wrong mbr cache.                                           |                                         | Minimum bounding rectangle of a face does not match minimum bounding box of the collection of edges reporting the face on their sides.                       |
| hole not in advertised face                            | Signed identifier of an edge, identifying the ring. See <a href="#">GetRingEdges</a> . |                                         | A ring of edges reporting a face on its exterior is contained in different face.                                                                             |



| Error                                     | id1                                                                                    | id2                                                                                    | Meaning                                                                                                                                                                                 |
|-------------------------------------------|----------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| not-isolated node has not-containing_face | Identifier of the ill-defined node.                                                    |                                                                                        | A node which is reported as being on the boundary of one or more edges is indicating a containing face.                                                                                 |
| isolated node has containing_face         | Identifier of the ill-defined node.                                                    |                                                                                        | A node which is not reported as being on the boundary of any edges is lacking the indication of a containing face.                                                                      |
| isolated node has wrong containing_face   | Identifier of the misrepresented node.                                                 |                                                                                        | A node which is not reported as being on the boundary of any edges indicates a containing face which is not the actual face containing it. See <a href="#">GetFaceContainingPoint</a> . |
| invalid next_right_edge                   | Identifier of the misrepresented edge.                                                 | Signed id of the edge which should be indicated as the next right edge.                | The edge indicated as the next edge encountered walking on the right side of an edge is wrong.                                                                                          |
| invalid next_left_edge                    | Identifier of the misrepresented edge.                                                 | Signed id of the edge which should be indicated as the next left edge.                 | The edge indicated as the next edge encountered walking on the left side of an edge is wrong.                                                                                           |
| mixed face labeling in ring               | Signed identifier of an edge, identifying the ring. See <a href="#">GetRingEdges</a> . |                                                                                        | Edges in a ring indicate conflicting faces on the walking side. This is also known as a "Side Location Conflict".                                                                       |
| non-closed ring                           | Signed identifier of an edge, identifying the ring. See <a href="#">GetRingEdges</a> . |                                                                                        | A ring of edges formed by following next_left_edge/next_right_edge attributes starts and ends on different nodes.                                                                       |
| face has multiple shells                  | Identifier of the contended face.                                                      | Signed identifier of an edge, identifying the ring. See <a href="#">GetRingEdges</a> . | More than a one ring of edges indicate the same face on its interior.                                                                                                                   |

Disponibilidad: 1.0.0

Mejorado: 2.0.0 detección de cruces de borde más eficiente y corrección de falsos positivos que existían en versiones anteriores.

Modificado: 2.2.0 los valores para id1 e id2 se intercambiaron para 'borde cruza nodo' para ser consistente con la descripción del error.

Changed: 3.2.0 added optional bbox parameter, perform face labeling and edge linking checks.

## Ejemplos

```
SELECT * FROM topology.ValidateTopology('ma_topo');
 error | id1 | id2
-----+-----+-----
face without edges | 1 |
```

**Ver también**

[validateTopology\\_returntype](#), [Topology\\_Load\\_Tiger](#)

### 8.3.9 ValidateTopologyRelation

ValidateTopologyRelation — Returns info about invalid topology relation records

**Synopsis**

setof record **ValidateTopologyRelation**(varchar toponame);

**Descripción**

Returns a set records giving information about invalidities in the relation table of the topology.

Availability: 3.2.0

**Ver también**

[ValidateTopology](#)

### 8.3.10 FindTopology

FindTopology — Returns a topology record by different means.

**Synopsis**

topology **FindTopology**(TopoGeometry topogeom);  
topology **FindTopology**(regclass layerTable, name layerColumn);  
topology **FindTopology**(name layerSchema, name layerTable, name layerColumn);  
topology **FindTopology**(text topoName);  
topology **FindTopology**(int id);

**Descripción**

Takes a topology identifier or the identifier of a topology-related object and returns a topology.topology record.

Availability: 3.2.0

**Ejemplos**

```
SELECT name(findTopology('features.land_parcel', 'feature'));
 name

city_data
(1 row)
```

**Ver también**

[FindLayer](#)

---

### 8.3.11 FindLayer

FindLayer — Returns a topology.layer record by different means.

#### Synopsis

```
topology.layer FindLayer(TopoGeometry tg);
topology.layer FindLayer(regclass layer_table, name feature_column);
topology.layer FindLayer(name schema_name, name table_name, name feature_column);
topology.layer FindLayer(integer topology_id, integer layer_id);
```

#### Descripción

Takes a layer identifier or the identifier of a topology-related object and returns a topology.layer record.

Availability: 3.2.0

#### Ejemplos

```
SELECT layer_id(findLayer('features.land_parcels', 'feature'));
 layer_id

 1
(1 row)
```

#### Ver también

[FindTopology](#)

## 8.4 Topology Statistics Management

Adding elements to a topology triggers many database queries for finding existing edges that will be split, adding nodes and updating edges that will node with the new linework. For this reason it is useful that statistics about the data in the topology tables are up-to-date.

PostGIS Topology population and editing functions do not automatically update the statistics because a updating stats after each and every change in a topology would be overkill, so it is the caller's duty to take care of that.



#### Note

That the statistics updated by autovacuum will NOT be visible to transactions which started before autovacuum process completed, so long-running transactions will need to run ANALYZE themselves, to use updated statistics.

## 8.5 Constructores de Topología

### 8.5.1 CreateTopology

CreateTopology — Creates a new topology schema and registers it in the topology.topology table.

## Synopsis

```
integer CreateTopology(varchar topology_schema_name);
integer CreateTopology(varchar topology_schema_name, integer srid);
integer CreateTopology(varchar topology_schema_name, integer srid, double precision prec);
integer CreateTopology(varchar topology_schema_name, integer srid, double precision prec, boolean hasz);
```

## Descripción

Creates a new topology schema with name `topology_name` and registers it in the `topology.topology` table. Topologies must be uniquely named. The topology tables (`edge_data`, `face`, `node`, and `relation`) are created in the schema. It returns the id of the topology.

The `srid` is the [spatial reference system](#) SRID for the topology.

The tolerance `prec` is measured in the units of the spatial reference system. The tolerance defaults to 0.

`hasz` defaults to false if not specified.

This is similar to the SQL/MM [ST\\_InitTopoGeo](#) but has more functionality.

Availability: 1.1

Enhanced: 2.0 added the signature accepting `hasZ`

## Ejemplos

Create a topology schema called `ma_topo` that stores edges and nodes in Massachusetts State Plane-meters (SRID = 26986). The tolerance represents 0.5 meters since the spatial reference system is meter-based.

```
SELECT topology.CreateTopology('ma_topo', 26986, 0.5);
```

Create a topology for Rhode Island called `ri_topo` in spatial reference system State Plane-feet (SRID = 3438)

```
SELECT topology.CreateTopology('ri_topo', 3438) AS topoid;
topoid

2
```

## Ver también

Section [4.5](#), [ST\\_InitTopoGeo](#), [Topology\\_Load\\_Tiger](#)

## 8.5.2 CopyTopology

**CopyTopology** — Makes a copy of a topology (nodes, edges, faces, layers and TopoGeometries) into a new schema

## Synopsis

```
integer CopyTopology(varchar existing_topology_name, varchar new_name);
```

## Descripción

Creates a new topology with name `new_name`, with SRID and precision copied from `existing_topology_name`. The nodes, edges and faces in `existing_topology_name` are copied into the new topology, as well as Layers and their associated TopoGeometries.



### Note

The new rows in the `topology.layer` table contain synthetic values for `schema_name`, `table_name` and `feature_column`. This is because the TopoGeometry objects exist only as a definition and are not yet available in a user-defined table.

Disponibilidad: 2.0.0

## Ejemplos

Make a backup of a topology called `ma_topo`.

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_backup');
```

## Ver también

Section [4.5](#), [CreateTopology](#), [RenameTopology](#)

## 8.5.3 ST\_InitTopoGeo

**ST\_InitTopoGeo** — Creates a new topology schema and registers it in the `topology.topology` table.

## Synopsis

text **ST\_InitTopoGeo**(varchar topology\_schema\_name);

## Descripción

This is the SQL-MM equivalent of [CreateTopology](#). It lacks options for spatial reference system and tolerance. It returns a text description of the topology creation, instead of the topology id.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo y Topo-Net 3: Detalles de la rutina: X.3.17

## Ejemplos

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation;
 astopocreation

Topology-Geometry 'topo_schema_to_create' (id:7) created.
```

## Ver también

[CreateTopology](#)

### 8.5.4 ST\_CreateTopoGeo

**ST\_CreateTopoGeo** — Agrega una colección de geometrías a una topología vacía dada y devuelve un mensaje que detalla el éxito.

#### Synopsis

text **ST\_CreateTopoGeo**(varchar atopoology, geometry acollection);

#### Descripción

Agrega una colección de geometrías a una topología vacía dada y devuelve un mensaje que detalla el éxito.

Útil para rellenar una topología vacía.

Disponibilidad: 2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de la rutina -- X.3.18

#### Ejemplos

```
-- Populate topology --
SELECT topology.ST_CreateTopoGeo('ri_topo',
 ST_GeomFromText('MULTILINESTRING((384744 236928,384750 236923,384769 236911,384799 236895,384811 236890,384833 236884,
 384844 236882,384866 236881,384879 236883,384954 236898,385087 236932,385117 236938,
 385167 236938,385203 236941,385224 236946,385233 236950,385241 236956,385254 236971,
 385260 236979,385268 236999,385273 237018,385273 237037,385271 237047,385267 237057,
 385225 237125,385210 237144,385192 237161,385167 237192,385162 237202,385159 237214,
 385159 237227,385162 237241,385166 237256,385196 237324,385209 237345,385234 237375,
 385237 237383,385238 237399,385236 237407,385227 237419,385213 237430,385193 237439,
 385174 237451,385170 237455,385169 237460,385171 237475,385181 237503,385190 237521,
 385200 237533,385206 237538,385213 237541,385221 237542,385235 237540,385242 237541,
 385249 237544,385260 237555,385270 237570,385289 237584,385292 237589,385291 237596,385284 237630))',3438)
);

 st_createtopogeo

Topology ri_topo populated

-- create tables and topo geometries --
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);

SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

#### Ver también

[AddTopoGeometryColumn](#), [CreateTopology](#), [DropTopology](#)

### 8.5.5 TopoGeo\_AddPoint

**TopoGeo\_AddPoint** — Agrega un punto a una topología existente utilizando una tolerancia y posiblemente dividiendo un borde existente.

**Synopsis**

integer **TopoGeo\_AddPoint**(varchar atopology, geometry apoint, float8 tolerance);

**Descripción**

Agrega un punto a una topología existente y devuelve su identificador. El punto dado se ajustará a los nodos o aristas existentes dentro de la tolerancia dada. Una arista existente se puede dividir por el punto de ajuste.

Disponibilidad: 2.0.0

**Ver también**

[TopoGeo\\_AddLineString](#), [TopoGeo\\_AddPolygon](#), [AddNode](#), [CreateTopology](#)

**8.5.6 TopoGeo\_AddLineString**

**TopoGeo\_AddLineString** — Agrega una cadena de línea a una topología existente utilizando una tolerancia y posiblemente dividiendo las aristas/caras existentes. Devuelve identificadores de borde.

**Synopsis**

SETOF integer **TopoGeo\_AddLineString**(varchar atopology, geometry aline, float8 tolerance);

**Descripción**

Agrega una cadena de línea a una topología existente y devuelve un conjunto de identificadores de borde que la forman. La línea dada se ajustará a los nodos o aristas existentes dentro de la tolerancia dada. Las aristas y caras existentes se pueden dividir por la línea.

**Note**

Updating statistics about topologies being loaded via this function is up to caller, see [maintaining statistics during topology editing and population](#).

Disponibilidad: 2.0.0

**Ver también**

[TopoGeo\\_AddPoint](#), [TopoGeo\\_AddPolygon](#), [AddEdge](#), [CreateTopology](#)

**8.5.7 TopoGeo\_AddPolygon**

**TopoGeo\_AddPolygon** — Agrega un polígono a una topología existente utilizando una tolerancia y posiblemente dividiendo las aristas/caras existentes. Devuelve identificadores de cara.

**Synopsis**

SETOF integer **TopoGeo\_AddPolygon**(varchar atopology, geometry apoly, float8 tolerance);

## Descripción

Agrega un polígono a una topología existente y devuelve un conjunto de identificadores de cara que lo forman. El límite del polígono dado se ajustará a los nodos o aristas existentes dentro de la tolerancia dada. Las aristas y caras existentes se pueden dividir por el límite del nuevo polígono.



### Note

Updating statistics about topologies being loaded via this function is up to caller, see [maintaining statistics during topology editing and population](#).

Disponibilidad: 2.0.0

## Ver también

[TopoGeo\\_AddPoint](#), [TopoGeo\\_AddLineString](#), [AddFace](#), [CreateTopology](#)

## 8.6 Editores de Topología

### 8.6.1 ST\_AddIsoNode

**ST\_AddIsoNode** — Agrega un nodo aislado a una cara de una topología y devuelve el identificador de nodo del nuevo nodo. Si la cara es nula, el nodo es creado de todas maneras.

## Synopsis

```
integer ST_AddIsoNode(varchar atopology, integer aface, geometry apoint);
```

## Descripción

Agrega un nodo aislado con la localización del punto `apoint` a una cara existente con identificador de cara `aface` a una topología `atopology` y devuelve el identificador de nodo de el nuevo nodo.

Si el sistema de referencia espacial (SRID) de la geometría de punto no es el mismo que el de la topología, el `apoint` no es una geometría de punto, el punto es nulo, o el punto intersecta un borde existente (incluso en los límites) entonces una excepción es lanzada. Si el punto ya existe como un nodo, se produce una excepción.

Si `aface` no es nulo y el `apoint` no está dentro de la cara, entonces una excepción es lanzada.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Net Rutinas: X+1.3.1

## Ejemplos

## Ver también

[AddNode](#), [CreateTopology](#), [DropTopology](#), [ST\\_Intersects](#)

### 8.6.2 ST\_AddIsoEdge

**ST\_AddIsoEdge** — Agrega un borde aislado definido por la geometría `alinestring` a una topología que conecta dos nodos aislados existentes `anode` y `anothernode` y devuelve el identificador de borde del nuevo borde.



## Synopsis

integer **ST\_AddIsoEdge**(varchar atopology, integer anode, integer anothernode, geometry alinestring);

## Descripción

Agrega un borde aislado definido por la geometría `alinestring` a una topología que conecta dos nodos aislados existentes `anode` y `anothernode` y devuelve el identificador de borde del nuevo borde.

Si el sistema de referencia espacial (SRID) de la geometría `alinestring` no es el mismo que la topología, cualquiera de los argumentos de entrada son nulos, o los nodos se contienen en más de una cara, o los nodos son el inicio o fin de los nodos de un borde existente, entonces una excepción es lanzada.

Si el `alinestring` no está dentro de la cara de la cara a la que pertenece `anode` y `anothernode`, entonces una excepción es lanzada.

Si el `anode` y `anothernode` no son los puntos de inicio y final de la `alinestring` entonces una excepción es lanzada.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de rutina: X.3.4

## Ejemplos

### Ver también

[ST\\_AddIsoNode](#), [ST\\_IsSimple](#), [ST\\_Within](#)

## 8.6.3 ST\_AddEdgeNewFaces

**ST\_AddEdgeNewFaces** — Agrega un nuevo borde y, si al hacerlo divide una cara, se elimina la cara original y es reemplazada con dos nuevas caras. Devuelve el identificador del borde recientemente agregado.

## Synopsis

integer **ST\_AddEdgeNewFaces**(varchar atopology, integer anode, integer anothernode, geometry acurve);

## Descripción

Agrega un nuevo borde y, si al hacerlo divide una cara, se elimina la cara original y es reemplazada con dos nuevas caras. Devuelve el identificador del borde recientemente agregado.

Actualiza todos los bordes unidos y relaciones en consecuencia existentes.

Si cualquier argumento es nulo, los nodos dados son desconocidos (ya deben existir en la tabla `node` del esquema de topología), el `acurve` no es un `LINESTRING`, el `anode` y `anothernode` no son el punto de inicio y final de `acurve` entonces un error es lanzado.

Si el sistema de referencia espacial (SRID) de la geometría `acurve` no es el mismo que la topología se lanza una excepción.

Disponibilidad: 2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.12

## Ejemplos

### Ver también

[ST\\_RemEdgeNewFace](#)

[ST\\_AddEdgeModFace](#)

### 8.6.4 ST\_AddEdgeModFace

**ST\_AddEdgeModFace** — Añada un nuevo borde y, si al hacerlo, divide una cara, modifica la cara original y añade una nueva cara.

#### Synopsis

integer **ST\_AddEdgeModFace**(varchar atopology, integer anode, integer anothernode, geometry acurve);

#### Descripción

Añade un nuevo borde y, si al hacerlo, se divide una cara, modifica la cara original y añade una nueva.



#### Note

Si es posible, la nueva cara se creará en el lado izquierdo del nuevo borde. Esto no será posible si la cara del lado izquierdo necesita ser Universe face (sin límites).

Devuelve el identificador del borde recientemente añadido.

Actualiza todos los bordes unidos y relaciones en consecuencia existentes.

Si cualquier argumento es nulo, los nodos dados son desconocidos (ya deben existir en la tabla `node` del esquema de topología), el `acurve` no es un `LINESTRING`, el `anode` y `anothernode` no son el punto de inicio y final de `acurve` entonces un error es lanzado.

Si el sistema de referencia espacial (SRID) de la geometría `acurve` no es el mismo que la topología se lanza una excepción.

Disponibilidad: 2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalle de Rutina: X.3.13

## Ejemplos

### Ver también

[ST\\_RemEdgeModFace](#)

[ST\\_AddEdgeNewFaces](#)

### 8.6.5 ST\_RemEdgeNewFace

**ST\_RemEdgeNewFace** — Elimina un borde y, si el borde eliminado separa dos caras, borra las caras originales y las reemplaza con una nueva cara.

#### Synopsis

integer **ST\_RemEdgeNewFace**(varchar atopology, integer anedge);

## Descripción

Elimina un borde y, si el borde eliminado separa dos caras, borra las caras originales y las reemplaza con una nueva cara.

Devuelve el identificador de una cara creada recientemente o NULL, si no se crea ninguna nueva cara. No se crea ninguna nueva cara cuando el borde eliminado está colgando o aislado o confinado con la cara del universo (posiblemente haciendo que el universo se inunde en la cara del otro lado).

Actualiza todos los bordes unidos y relaciones en consecuencia existentes.

Se niega a eliminar un borde que participa en la definición de un TopoGeometry existente. Se niega a sanear dos caras si cualquier TopoGeometry es definido por sólo uno de ellos (y no el otro).

Si algún argumento es null, se desconoce el borde dado (debe existir ya en la tabla `edge` del esquema de topología), el nombre de la topología no es válido entonces se produce un error.

Disponibilidad: 2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.14

## Ejemplos

### Ver también

[ST\\_RemEdgeModFace](#)

[ST\\_AddEdgeNewFaces](#)

## 8.6.6 ST\_RemEdgeModFace

**ST\_RemEdgeModFace** — Removes an edge, and if the edge separates two faces deletes one face and modifies the other face to cover the space of both.

## Synopsis

integer **ST\_RemEdgeModFace**(varchar atopology, integer anedge);

## Descripción

Removes an edge, and if the removed edge separates two faces deletes one face and modifies the other face to cover the space of both. Preferentially keeps the face on the right, to be consistent with [ST\\_AddEdgeModFace](#). Returns the id of the face which is preserved.

Actualiza todos los bordes unidos y relaciones en consecuencia existentes.

Se niega a eliminar un borde que participa en la definición de un TopoGeometry existente. Se niega a sanear dos caras si cualquier TopoGeometry es definido por sólo uno de ellos (y no el otro).

Si algún argumento es null, se desconoce el borde dado (debe existir ya en la tabla `edge` del esquema de topología), el nombre de la topología no es válido entonces se produce un error.

Disponibilidad: 2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.15

## Ejemplos

### Ver también

[ST\\_AddEdgeModFace](#)

[ST\\_RemEdgeNewFace](#)

### 8.6.7 ST\_ChangeEdgeGeom

ST\_ChangeEdgeGeom — Cambia la forma de un borde sin afectar la estructura de la topología.

#### Synopsis

integer **ST\_ChangeEdgeGeom**(varchar atopology, integer anedge, geometry acurve);

#### Descripción

Cambia la forma de un borde sin afectar la estructura de la topología.

If any arguments are null, the given edge does not exist in the `edge` table of the topology schema, the `acurve` is not a `LINESTRING`, or the modification would change the underlying topology then an error is thrown.

Si el sistema de referencia espacial (SRID) de la geometría `acurve` no es el mismo que la topología se lanza una excepción.

Si el nuevo `acurve` no es simple, entonces un error es lanzado.

Si al mover el borde de la vieja a la nueva posición golpease un obstáculo entonces se produce un error.

Disponibilidad: 1.1.0

Mejorado: 2.0.0 agrega aplicación de consistencia topológica



This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalle de Rutina X.3.6

#### Ejemplos

```
SELECT topology.ST_ChangeEdgeGeom('ma_topo', 1,
 ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6 893844.3,227641.6
 893816.6, 227704.5 893778.5)', 26986));

Edge 1 changed
```

#### Ver también

[ST\\_AddEdgeModFace](#)

[ST\\_RemEdgeModFace](#)

[ST\\_ModEdgeSplit](#)

### 8.6.8 ST\_ModEdgeSplit

ST\_ModEdgeSplit — Dividir un borde creando un nuevo nodo a lo largo de un borde existente, modificando el borde original y agregando un nuevo borde.

#### Synopsis

integer **ST\_ModEdgeSplit**(varchar atopology, integer anedge, geometry apoint);

## Descripción

Dividir un borde creando un nuevo nodo a lo largo de un borde existente, modificando el borde original y agregando un nuevo borde. Actualiza todos los bordes unidos existentes y relaciones en consecuencia. Devuelve el identificador del nodo recientemente agregado.

Availability: 1.1

Cambiado: 2.0 - En versiones anteriores, esto fue mal llamado ST\_ModEdgesSplit



This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.9

## Ejemplos

```
-- Add an edge --
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227592 893910, 227600 893910)', 26986)) As edgeid;

-- edgeid-
3

-- Split the edge --
SELECT topology.ST_ModEdgeSplit('ma_topo', 3, ST_SetSRID(ST_Point(227594,893910),26986)) As node_id;
 node_id

7
```

## Ver también

[ST\\_NewEdgesSplit](#), [ST\\_ModEdgeHeal](#), [ST\\_NewEdgeHeal](#), [AddEdge](#)

## 8.6.9 ST\_ModEdgeHeal

ST\_ModEdgeHeal — Cura dos aristas eliminando el nodo que las conecta, modificando la primera arista y eliminando la segunda arista. Devuelve el identificador del nodo eliminado.

## Synopsis

```
int ST_ModEdgeHeal(varchar atopology, integer anedge, integer anotheredge);
```

## Descripción

Cura dos aristas eliminando el nodo que las conecta, modificando la primera arista y eliminando la segunda arista. Devuelve el identificador del nodo eliminado. Actualiza todos los bordes y relaciones unidos existentes en consecuencia.

Disponibilidad: 2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.9

## Ver también

[ST\\_ModEdgeSplit](#) [ST\\_NewEdgesSplit](#)

### 8.6.10 ST\_NewEdgeHeal

**ST\_NewEdgeHeal** — Cura dos aristas eliminando el nodo que las conecta, eliminando ambas aristas y sustituyéndolas por una arista cuya dirección sea la misma que la primera arista proporcionada.

#### Synopsis

```
int ST_NewEdgeHeal(varchar atopology, integer anedge, integer anotheredge);
```

#### Descripción

Cura dos aristas eliminando el nodo que las conecta, eliminando ambas aristas y sustituyéndolas por una arista cuya dirección sea la misma que la primera arista proporcionada. Devuelve el identificador de la nueva aristas reemplazante de las curadas. Actualiza todas las aristas y relaciones unidos existentes en consecuencia.

Disponibilidad: 2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.9

#### Ver también

[ST\\_ModEdgeHeal](#) [ST\\_ModEdgeSplit](#) [ST\\_NewEdgesSplit](#)

### 8.6.11 ST\_MoveIsoNode

**ST\_MoveIsoNode** — Mueve un nodo aislado en una topología de un punto a otro. Si la nueva geometría `apoint` existe como nodo se lanza un error. Devuelve la descripción del movimiento.

#### Synopsis

```
text ST_MoveIsoNode(varchar atopology, integer anode, geometry apoint);
```

#### Descripción

Mueve un nodo aislado en una topología de un punto a otro. Si la nueva geometría `apoint` existe como un nodo un error es lanzado.

If any arguments are null, the `apoint` is not a point, the existing node is not isolated (is a start or end point of an existing edge), new node location intersects an existing edge (even at the end points) or the new location is in a different face (since 3.2.0) then an exception is thrown.

Si el sistema de referencia espacial (SRID) de la geometría de punto no es el mismo que el de la topología se lanza una excepción.

Disponibilidad: 2.0.0

Enhanced: 3.2.0 ensures the nod cannot be moved in a different face



This method implements the SQL/MM specification. SQL-MM: Topo-Net Rutina: X.3.2

---

## Ejemplos

```
-- Add an isolated node with no face --
SELECT topology.ST_AddIsoNode('ma_topo', NULL, ST_GeomFromText('POINT(227579 893916)', 26986)) As nodeid;
nodeid

7
-- Move the new node --
SELECT topology.ST_MoveIsoNode('ma_topo', 7, ST_GeomFromText('POINT(227579.5 893916.5)', 26986)) As descrip;
descrip

Isolated Node 7 moved to location 227579.5,893916.5
```

## Ver también

[ST\\_AddIsoNode](#)

### 8.6.12 ST\_NewEdgesSplit

**ST\_NewEdgesSplit** — Divide un borde creando un nuevo nodo a lo largo de un borde existente, eliminando el borde original y reemplazándolo con dos bordes nuevos. Devuelve el identificador del nuevo nodo creado que une los nuevos bordes.

## Synopsis

integer **ST\_NewEdgesSplit**(varchar atopology, integer anedge, geometry apoint);

## Descripción

Divide un borde con el identificador de borde `anedge` creando un nodo nuevo con la localización del punto `apoint` a lo largo del borde actual, eliminando el borde original y reemplazando con dos bordes nuevos. Devuelve el identificador del nuevo nodo creado que une los nuevos bordes. Actualiza todos los bordes unidos existentes y relaciones en consecuencia.

Si el sistema de referencia espacial (SRID) de la geometría de punto no es el mismo que el de la topología, el `apoint` no es una geometría de punto, el punto es nulo, el punto ya existe como un nodo, el borde no corresponde a un borde existente o el punto no está dentro del borde entonces se lanza una excepción.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Net Rutina: X.3.8

## Ejemplos

```
-- Add an edge --
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575 893917,227592 893900)', 26986)) As edgeid;
-- result-
edgeid

2
-- Split the new edge --
SELECT topology.ST_NewEdgesSplit('ma_topo', 2, ST_GeomFromText('POINT(227578.5 893913.5)', 26986)) As newnodeid;
newnodeid

6
```

**Ver también**

[ST\\_ModEdgeSplit](#) [ST\\_ModEdgeHeal](#) [ST\\_NewEdgeHeal](#) [AddEdge](#)

**8.6.13 ST\_RemoveIsoNode**

**ST\_RemoveIsoNode** — Elimina un nodo aislado y devuelve la descripción de la acción. Si el nodo no está aislado (es el inicio o el final de un borde), entonces se lanza una excepción.

**Synopsis**

text **ST\_RemoveIsoNode**(varchar atopology, integer anode);

**Descripción**

Elimina un nodo aislado y devuelve la descripción de la acción. Si el nodo no está aislado (es el inicio o el final de un borde), entonces se lanza una excepción.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X+1.3.3

**Ejemplos**

```
-- Remove an isolated node with no face --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7) As result;
 result

Isolated node 7 removed
```

**Ver también**

[ST\\_AddIsoNode](#)

**8.6.14 ST\_RemoveIsoEdge**

**ST\_RemoveIsoEdge** — Elimina un borde aislado y devuelve la descripción de la acción. Si el borde no está aislado, se lanza una excepción.

**Synopsis**

text **ST\_RemoveIsoEdge**(varchar atopology, integer anedge);

**Descripción**

Elimina un borde aislado y devuelve la descripción de la acción. Si el borde no está aislado, se lanza una excepción.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X+1.3.3



## Ejemplos

```
-- Remove an isolated node with no face --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7) As result;
 result

Isolated node 7 removed
```

## Ver también

[ST\\_AddIsoNode](#)

## 8.7 Accesorios de Topología

### 8.7.1 GetEdgeByPoint

**GetEdgeByPoint** — Busca el identificador de arista de una arista que cruza un punto determinado.

#### Synopsis

integer **GetEdgeByPoint**(varchar atopology, geometry apoint, float8 toll);

#### Descripción

Recupera el identificador de una arista que cruza con un point.

La función devuelve un entero (identificador de borde) dada una topología, un POINT y una tolerancia. Si la tolerancia = 0 el punto tiene que intersectar el borde.

Si un apoint no intersecta una arista, devuelve 0 (cero).

Si se usa tolerancia > 0 y hay más de un borde cerca del punto, entonces se lanza una excepción.



#### Note

Si la tolerancia = 0, la función utiliza ST\_Intersects de otra manera utiliza ST\_DWithin.

Realizado por el módulo GEOS.

Disponibilidad: 2.0.0

## Ejemplos

Estos ejemplos utilizan los bordes que hemos creado en [AddEdge](#)

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As withlmtol, topology.GetEdgeByPoint(' ←
 ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('SRID=26986;POINT(227622.6 893843)') As geom;
 withlmtol | withnotol
-----+-----
 |
2 | | 0
```

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

-- get error --
ERROR: Two or more edges found
```

## Ver también

[AddEdge](#), [GetNodeByPoint](#), [GetFaceByPoint](#)

## 8.7.2 GetFaceByPoint

GetFaceByPoint — Finds face intersecting a given point.

### Synopsis

integer **GetFaceByPoint**(varchar atopology, geometry apoint, float8 toll);

### Descripción

Finds a face referenced by a Point, with given tolerance.

The function will effectively look for a face intersecting a circle having the point as center and the tolerance as radius.

If no face intersects the given query location, 0 is returned (universal face).

If more than one face intersect the query location an exception is thrown.

Disponibilidad: 2.0.0

Enhanced: 3.2.0 more efficient implementation and clearer contract, stops working with invalid topologies.

### Ejemplos

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 10) As with1mtol, topology.GetFaceByPoint('ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('POINT(234604.6 899382.0)') As geom;

with1mtol | withnotol
-----+-----
1 | 0
```

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('POINT(227591.9 893900.4)') As geom;

-- get error --
ERROR: Two or more faces found
```

## Ver también

[GetFaceContainingPoint](#), [AddFace](#), [GetNodeByPoint](#), [GetEdgeByPoint](#)

## 8.7.3 GetFaceContainingPoint

GetFaceContainingPoint — Finds the face containing a point.

## Synopsis

integer **GetFaceContainingPoint**(text atopology, geometry apoint);

## Descripción

Returns the id of the face containing a point.

An exception is thrown if the point falls on a face boundary.



### Note

The function relies on a valid topology, using edge linking and face labeling.

---

Availability: 3.2.0

## Ver también

[ST\\_GetFaceGeometry](#)

## 8.7.4 GetNodeByPoint

GetNodeByPoint — Busca el identificador de nodo de un nodo en un punto de ubicación.

## Synopsis

integer **GetNodeByPoint**(varchar atopology, geometry apoint, float8 tol1);

## Descripción

Recupera el identificador de un nodo en un punto de ubicación.

La función devuelve un entero (identificador de nodo) dada una topología, un POINT y una tolerancia. Si la tolerancia es 0 significa intersección exacta, de lo contrario recupera el nodo de un intervalo.

Si `apoint` no intersecta un nodo, devuelve 0 (cero).

Si utiliza tolerancia  $> 0$  y hay más de un nodo cerca del punto, se produce una excepción.



### Note

Si la tolerancia = 0, la función utiliza ST\_Intersects de otra manera utiliza ST\_DWithin.

---

Realizado por el módulo GEOS.

Disponibilidad: 2.0.0

---

## Ejemplos

Estos ejemplos utilizan los bordes que hemos creado en [AddEdge](#)

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
nearnode

2
```

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1000) As too_much_tolerance
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

----get error--
ERROR: Two or more nodes found
```

## Ver también

[AddEdge](#), [GetEdgeByPoint](#), [GetFaceByPoint](#)

## 8.7.5 GetTopologyID

**GetTopologyID** — Devuelve el identificador de una topología en la tabla topology.topology dado el nombre de la topología.

### Synopsis

integer **GetTopologyID**(varchar toponame);

### Descripción

Devuelve el identificador de una topología en la tabla topology.topology dado el nombre de la topología.

Availability: 1.1

## Ejemplos

```
SELECT topology.GetTopologyID('ma_topo') As topo_id;
topo_id

1
```

## Ver también

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologySRID](#)

## 8.7.6 GetTopologySRID

**GetTopologySRID** — Devuelve el SRID de una topología en la tabla topology.topology dado el nombre de la topología.

### Synopsis

integer **GetTopologyID**(varchar toponame);

### Descripción

Devuelve el identificador del sistema de referencia de una topología en la tabla topology.topology dado el nombre de la topología.

Disponibilidad: 2.0.0

### Ejemplos

```
SELECT topology.GetTopologySRID('ma_topo') As SRID;
SRID

4326
```

### Ver también

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologyID](#)

## 8.7.7 GetTopologyName

GetTopologyName — Devuelve el nombre de una topología (esquema) dado el identificador de la topología.

### Synopsis

varchar **GetTopologyName**(integer topology\_id);

### Descripción

Devuelve el nombre de topología (esquema) de una topología de la tabla topology.topology dado el identificador de topología de la topología.

Availability: 1.1

### Ejemplos

```
SELECT topology.GetTopologyName(1) As topo_name;
topo_name

ma_topo
```

### Ver también

[CreateTopology](#), [DropTopology](#), [GetTopologyID](#), [GetTopologySRID](#)

## 8.7.8 ST\_GetFaceEdges

ST\_GetFaceEdges — Devuelve un conjunto de bordes ordenados que ligan aface.

### Synopsis

getfaceedges\_returntype **ST\_GetFaceEdges**(varchar atopology, integer aface);

---

## Descripción

Devuelve un conjunto de bordes ordenados que ligan a `face`. Cada salida consta de una secuencia e identificador de borde. Los números de secuencia comienzan con el valor 1.

La enumeración de los bordes de cada anillo comienza desde el borde con el identificador más pequeño. El orden de los bordes sigue la regla de la izquierda (la cara enmarcada está a la izquierda de cada borde dirigido).

Disponibilidad: 2.0



This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.5

## Ejemplos

```
-- Returns the edges bounding face 1
SELECT (topology.ST_GetFaceEdges('tt', 1)).*;
-- result --
sequence | edge
-----+-----
1 | -4
2 | 5
3 | 7
4 | -6
5 | 1
6 | 2
7 | 3
(7 rows)
```

```
-- Returns the sequence, edge id
-- and geometry of the edges that bound face 1
-- If you just need geom and seq, can use ST_GetFaceGeometry
SELECT t.seq, t.edge, geom
FROM topology.ST_GetFaceEdges('tt',1) As t(seq,edge)
INNER JOIN tt.edge AS e ON abs(t.edge) = e.edge_id;
```

## Ver también

[GetRingEdges](#), [AddFace](#), [ST\\_GetFaceGeometry](#)

## 8.7.9 ST\_GetFaceGeometry

`ST_GetFaceGeometry` — Devuelve el polígono en la topología dada con el identificador de la cara especificada.

## Synopsis

geometry **ST\_GetFaceGeometry**(varchar atopology, integer aface);

## Descripción

Devuelve el polígono en la topología dada con el identificador de cara especificado. Construye el polígono de los bordes que componen la cara.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.16

## Ejemplos

```
-- Returns the wkt of the polygon added with AddFace
SELECT ST_AsText(topology.ST_GetFaceGeometry('ma_topo', 1)) As facegeomwkt;
-- result --
 facegeomwkt

POLYGON((234776.9 899563.7,234896.5 899456.7,234914 899436.4,234946.6 899356.9,
234872.5 899328.7,234891 899285.4,234992.5 899145,234890.6 899069,
234755.2 899255.4,234612.7 899379.4,234776.9 899563.7))
```

## Ver también

[AddFace](#)

### 8.7.10 GetRingEdges

**GetRingEdges** — Devuelve el conjunto ordenado de identificadores de borde con signo asignado al caminar en un lado de borde dado.

#### Synopsis

```
getfaceedges_returntype GetRingEdges(varchar atopology, integer aring, integer max_edges=null);
```

#### Descripción

Devuelve el conjunto ordenado de identificadores de borde con signo asignado al caminar en un lado de borde dado. Cada salida consta de una secuencia y un identificador de borde con signo. Los números de secuencia comienzan con el valor 1.

Si pasa un identificador de borde positivo, la caminata comienza en el lado izquierdo del borde correspondiente y sigue la dirección del borde. Si pasa un identificador de borde negativo, el paseo comienza en el lado derecho de la misma y retrocede.

Si `max_edges` no es nulo no más que esos registros son devueltos por esa función. Esto se supone que es un parámetro de seguridad cuando se trata de topologías posiblemente inválidas.



#### Note

Esta función utiliza metadatos enlazados de anillo de borde.

Disponibilidad: 2.0.0

## Ver también

[ST\\_GetFaceEdges](#), [GetNodeEdges](#)

### 8.7.11 GetNodeEdges

**GetNodeEdges** — Devuelve un conjunto ordenado de aristas incidente al nodo dado.

## Synopsis

`getfaceedges_returntype` **GetNodeEdges**(varchar atopology, integer anode);

## Descripción

Devuelve un conjunto ordenado de aristas incidente al nodo dado. Cada salida consta de una secuencia y un identificador de arista con signo. Los números de secuencia comienzan con el valor 1. Una arista positiva comienza en el nodo dado. Una arista negativa termina en el nodo dado. Las aristas cerradas aparecerán dos veces (con ambos signos). El orden es en sentido horario empezando desde el norte.



### Note

Esta función calcula pedidos en lugar de derivar de metadatos y es así útil para construir el bode de anillo de vinculación.

Disponibilidad: 2.0

## Ver también

`getfaceedges_returntype`, `GetRingEdges`, `ST_Azimuth`

## 8.8 Procesamiento de Topología

### 8.8.1 Polygonize

Polygonize — Busca y registra todas las caras definidas por aristas de topología.

## Synopsis

text **Polygonize**(varchar toponame);

## Descripción

Registra todas las caras que se pueden crear como primitivas de borde de topología.

Se supone que la topología de destino no contiene bordes que se intersectan a sí mismos.



### Note

Se reconocen caras ya conocidas, por lo que es seguro llamar a Polygonize varias veces en la misma topología.



### Note

Esta función no utiliza ni establece los campos `next_left_edge` y `next_right_edge` de la tabla `Edge`.

Disponibilidad: 2.0.0



**Ver también**[AddFace](#), [ST\\_Polygonize](#)**8.8.2 AddNode**

**AddNode** — Agrega un nodo de punto a la tabla de nodos del esquema de topología especificado y devuelve el identificador de nodo del nuevo nodo. Si el punto ya existe como nodo, se devuelve el identificador de nodo existente.

**Synopsis**

```
integer AddNode(varchar toponame, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);
```

**Descripción**

Agrega un nodo de punto a la tabla de nodos en el esquema de topología especificado. La función [AddEdge](#) agrega automáticamente los puntos de inicio y fin de una arista cuando se le llama por lo tanto no es necesario agregar nodos de arista de forma explícita.

Si se encuentra alguna arista que cruza el nodo, se produce una excepción o se divide la arista, dependiendo del valor del parámetro `allowEdgeSplitting`.

Si `computeContainingFace` es verdadero un nodo recién añadido obtendrá la cara de contención correcta calculada.

**Note**

Si la geometría `apoint` ya existe como nodo, el nodo no se agrega pero se devuelve el identificador del nodo existente.

Disponibilidad: 2.0.0

**Ejemplos**

```
SELECT topology.AddNode('ma_topo', ST_GeomFromText('POINT(227641.6 893816.5)', 26986)) As nodeid;
-- result --
nodeid

4
```

**Ver también**[AddEdge](#), [CreateTopology](#)**8.8.3 AddEdge**

**AddEdge** — Agrega una arista de `LineString` a la tabla de arista y los puntos de inicio y fin asociados a la tabla nodos de puntos del esquema de topología usando la geometría `LineString` especificada y devuelve el identificador de arista de la arista nueva (o existente).

## Synopsis

integer **AddEdge**(varchar toponame, geometry aline);

## Descripción

Agrega un borde a la tabla arista y los nodos asociados a la tabla nodos del esquema `toponame` especificado usando la geometría `LineString` especificada y devuelve el identificador de arista del registro nuevo o existente. El borde recién añadido tiene la cara de "universo" en ambos lados y enlaces a sí mismo.



### Note

Si la geometría `aline` se cruza, se superpone, contiene o está contenida por un borde de cadena de línea existente, entonces se genera un error y no se agrega el borde.



### Note

La geometría de `aline` debe tener el mismo `srid` que el definido para la topología de lo contrario se lanzará un error de sistema de referencia espacial no válido.

Realizado por el módulo GEOS.

Disponibilidad: 2.0.0

## Ejemplos

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575.8 893917.2,227591.9 893900.4)', 26986)) As edgeid;
-- result-
edgeid

1

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6 893844.2,227641.6 893816.5, 227704.5 893778.5)', 26986)) As edgeid;
-- result --
edgeid

2

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.2 893900, 227591.9 893900.4, 227704.5 893778.5)', 26986)) As edgeid;
-- gives error --
ERROR: Edge intersects (not on endpoints) with existing edge 1
```

## Ver también

[TopoGeo\\_AddLineString](#), [CreateTopology](#), [Section 4.5](#)

## 8.8.4 AddFace

**AddFace** — Registra una primitiva de cara a una topología y obtiene su identificador.

## Synopsis

integer **AddFace**(varchar toponame, geometry apolygon, boolean force\_new=false);

## Descripción

Registra una primitiva de cara a una topología y obtiene su identificador.

Para una cara recién agregada, los bordes que forman sus límites y los contenidos en la cara se actualizarán para tener valores correctos en los campos `left_face` y `right_face`. Los nodos aislados contenidos en la cara también se actualizarán para tener un valor del campo `containing_face`.



### Note

Esta función no utiliza ni establece los campos `next_left_edge` y `next_right_edge` de la tabla `Edge`.

Se supone que la topología de destino es válida (que no contiene aristas de intersección). Se plantea una excepción si: el límite del polígono no está definido completamente por los bordes existentes o el polígono se superpone con una cara existente.

Si la geometría `apolygon` ya existe como cara, entonces: si `force_new` es falso (el valor predeterminado) se devuelve el id de cara de la cara existente; si `force_new` es verdadero; , se asignará un nuevo identificador a la cara recién registrada.



### Note

Cuando se realiza un nuevo registro de una cara existente (`force_new=true`), no se tomará ninguna acción para resolver las referencias pendientes a la cara existente en la arista, nodo y a tablas de relación, ni se actualizará el campo `MBR` del registro de cara existente. Depende de la persona que llama hacerse cargo de ello.



### Note

La geometría `apolygon` debe tener el mismo `srid` que el definido para la topología de lo contrario se lanzará un error de sistema de referencia espacial no válido.

Disponibilidad: 2.0.0

## Ejemplos

```
-- first add the edges we use generate_series as an iterator (the below
-- will only work for polygons with < 10000 points because of our max in gs)
SELECT topology.AddEdge('ma_topo', ST_MakeLine(ST_PointN(geom,i), ST_PointN(geom, i + 1))) ←
 As edgeid
FROM (SELECT ST_NPoints(geom) AS npt, geom
 FROM
 (SELECT ST_Boundary(ST_GeomFromText('POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5 899328.7,
 234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
 234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986)) As geom
) As geoms) As facen CROSS JOIN generate_series(1,10000) As i
 WHERE i < npt;
-- result --
edgeid

 3
 4
```

```

5
6
7
8
9
10
11
12
(10 rows)
-- then add the face -

SELECT topology.AddFace('ma_topo',
 ST_GeomFromText('POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5 ←
 899328.7,
 234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
 234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986)) As faceid;
-- result --
faceid

1

```

**Ver también**

[AddEdge](#), [CreateTopology](#), [Section 4.5](#)

**8.8.5 ST\_Simplify**

**ST\_Simplify** — Devuelve una versión "simplificada" de la geometría de la TopoGeometry dada usando el algoritmo de Douglas-Peucker.

**Synopsis**

geometry **ST\_Simplify**(TopoGeometry tg, float8 tolerance);

**Descripción**

Devuelve una versión "simplificada" de la geometría de la TopoGeometry dada usando el algoritmo de Douglas-Peucker en cada arista componente.

**Note**

La geometría devuelta puede ser no simple o no válida.  
Dividir las aristas de los componentes puede ayudar a conservar la simplicidad/validez.

Realizado por el módulo GEOS.

Disponibilidad: 2.1.0

**Ver también**

Geometría [ST\\_Simplify](#), [ST\\_IsSimple](#), [ST\\_IsValid](#), [ST\\_ModEdgeSplit](#)

### 8.8.6 RemoveUnusedPrimitives

RemoveUnusedPrimitives — Removes topology primitives which not needed to define existing TopoGeometry objects.

#### Synopsis

```
int RemoveUnusedPrimitives(text topology_name, geometry bbox);
```

#### Descripción

Finds all primitives (nodes, edges, faces) that are not strictly needed to represent existing TopoGeometry objects and removes them, maintaining topology validity (edge linking, face labeling) and TopoGeometry space occupation.

No new primitive identifiers are created, but rather existing primitives are expanded to include merged faces (upon removing edges) or healed edges (upon removing nodes).

Availability: 3.3.0

#### Ver también

[ST\\_ModEdgeHeal](#), [ST\\_RemEdgeModFace](#)

## 8.9 Constructores de Geometría Topográfica

### 8.9.1 CreateTopoGeom

CreateTopoGeom — Crea un nuevo objeto de geometría topo de la matriz de elementos topo - tg\_type: 1:[multi]point, 2:[multi]line, 3:[multi]poly, 4:collection

#### Synopsis

```
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id, topoelementarray tg_objs);
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id);
```

#### Descripción

Crea un objeto de topogeometría para la capa indicada por `layer_id` y lo registra en la tabla `relations` del esquema `toponame`.

`tg_type` es un entero: 1:[multi]point (puncal), 2:[multi]line (lineal), 3:[multi]poly (areal), 4:collection. `layer_id` es el identificador de capa de la tabla `topology.layer`.

las capas puntuales se forman a partir de un conjunto de nodos, las capas lineales se forman a partir de un conjunto de aristas, las capas de área se forman a partir de un conjunto de caras, y las colecciones se pueden formar a partir de una mezcla de nodos, aristas y caras.

Omitir la matriz de componentes genera un objeto TopoGeometry vacío.

Availability: 1.1

#### Ejemplos: formulario de aristas existentes

Crea un topogeom en el esquema `ri_topo` para la capa 2 (nuestra `ri_roads`), de tipo (2) LINE, para el primer borde (cargamos en `ST_CreateTopoGeo`).

```
INSERT INTO ri.ri_roads(road_name, topo) VALUES('Unknown', topology.CreateTopoGeom('ri_topo' ←
 ',2,2, '{{1,2}}'::topology.topoelementarray);
```

### Ejemplos: Convierte una geometría de área a una topogeometría

Digamos que tenemos geometrías que deben ser formadas a partir de una colección de caras. Tenemos por ejemplo la tabla `blockgroups` y queremos conocer la geometría topo de cada grupo de bloques. Si nuestros datos estuvieran perfectamente alineados, podríamos hacer esto:

```
-- create our topo geometry column --
SELECT topology.AddTopoGeometryColumn(
 'topo_boston',
 'boston', 'blockgroups', 'topo', 'POLYGON');

-- addtopogeometrycolumn --
1

-- update our column assuming
-- everything is perfectly aligned with our edges
UPDATE boston.blockgroups AS bg
 SET topo = topology.CreateTopoGeom('topo_boston'
 ,3,1
 , foo.bfaces)
FROM (SELECT b.gid, topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
 FROM boston.blockgroups As b
 INNER JOIN topo_boston.face As f ON b.geom && f.mbr
 WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
 GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;
```

```
--the world is rarely perfect allow for some error
--count the face if 50% of it falls
-- within what we think is our blockgroup boundary
UPDATE boston.blockgroups AS bg
 SET topo = topology.CreateTopoGeom('topo_boston'
 ,3,1
 , foo.bfaces)
FROM (SELECT b.gid, topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
 FROM boston.blockgroups As b
 INNER JOIN topo_boston.face As f ON b.geom && f.mbr
 WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
 OR
 (ST_Intersects(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
 AND ST_Area(ST_Intersection(b.geom, topology.ST_GetFaceGeometry('topo_boston', ←
 f.face_id))) >
 ST_Area(topology.ST_GetFaceGeometry('topo_boston', f.face_id))*0.5
)
 GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;

-- and if we wanted to convert our topogeometry back
-- to a denormalized geometry aligned with our faces and edges
-- cast the topo to a geometry
-- The really cool thing is my new geometries
-- are now aligned with my tiger street centerlines
UPDATE boston.blockgroups SET new_geom = topo::geometry;
```

### Ver también

[AddTopoGeometryColumn](#), [toTopoGeom](#) [ST\\_CreateTopoGeo](#), [ST\\_GetFaceGeometry](#), [TopoElementArray](#), [TopoElementArray\\_Agg](#)

## 8.9.2 toTopoGeom

toTopoGeom — Convierte un Geometry simple en una geometría topo.

### Synopsis

```
topogeometry toTopoGeom(geometry geom, varchar toponame, integer layer_id, float8 tolerance);
topogeometry toTopoGeom(geometry geom, topogeometry topogeom, float8 tolerance);
```

### Descripción

Convierte una geometría simple en un **TopoGeometry**.

Las primitivas topológicas necesarias para representar la geometría de entrada se añadirán a la topología subyacente, posiblemente dividiendo los existentes, y se asociarán con la salida TopoGeometry en la tabla *relation*.

Los objetos TopoGeometry existentes (con la posible excepción de *topogeoms* si se les da) conservarán sus formas.

Cuando se da la *tolerance* se usará para ajustar la geometría de entrada a las primitivas existentes.

En la primera forma se creará un nuevo TopoGeometry para la capa dada (*layer\_id*) de la topología dada (*toponame*).

En la segunda forma las primitivas resultantes de la conversión se añadirán a la TopoGeometry preexistente (*topogeom*), añadiendo posiblemente espacio a su forma final. Para que la nueva forma Reemplace completamente la antigua ver **clearTopoGeom**.

Disponibilidad: 2.0

Mejorado: 2.1.0 agrega la versión tomando una TopoGeometry existente.

### Ejemplos

Se trata de un flujo de trabajo autónomo completo

```
-- do this if you don't have a topology setup already
-- creates topology not allowing any tolerance
SELECT topology.CreateTopology('topo_boston_test', 2249);
-- create a new table
CREATE TABLE nei_topo(gid serial primary key, nei varchar(30));
--add a topogeometry column to it
SELECT topology.AddTopoGeometryColumn('topo_boston_test', 'public', 'nei_topo', 'topo', ' ←
MULTIPOLYGON') As new_layer_id;
new_layer_id

1

--use new layer id in populating the new topogeometry column
-- we add the topogeoms to the new layer with 0 tolerance
INSERT INTO nei_topo(nei, topo)
SELECT nei, topology.toTopoGeom(geom, 'topo_boston_test', 1)
FROM neighborhoods
WHERE gid BETWEEN 1 and 15;

--use to verify what has happened --
SELECT * FROM
 topology.TopologySummary('topo_boston_test');

-- summary--
Topology topo_boston_test (5), SRID 2249, precision 0
61 nodes, 87 edges, 35 faces, 15 topogeoms in 1 layers
Layer 1, type Polygonal (3), 15 topogeoms
Deploy: public.nei_topo.topo
```

```
-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);

-- Get the no-one-lands left by the above operation
-- I think GRASS calls this "polygon0 layer"
SELECT ST_GetFaceGeometry('topo_boston_test', f.face_id)
 FROM topo_boston_test.face f
 WHERE f.face_id
> 0 -- don't consider the universe face
 AND NOT EXISTS (-- check that no TopoGeometry references the face
 SELECT * FROM topo_boston_test.relation
 WHERE layer_id = 1 AND element_id = f.face_id
);
```

#### Ver también

[CreateTopology](#), [AddTopoGeometryColumn](#), [CreateTopoGeom](#), [TopologySummary](#), [clearTopoGeom](#)

### 8.9.3 TopoElementArray\_Agg

`TopoElementArray_Agg` — Devuelve un `topoelementarray` para un conjunto de `element_id`, matriz de tipo (`topoelements`).

#### Synopsis

`topoelementarray` **`TopoElementArray_Agg`**(`topoelement set tefield`);

#### Descripción

Usado para crear una [TopoElementArray](#) desde un conjunto de [TopoElement](#).

Disponibilidad: 2.0.0

#### Ejemplos

```
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
 FROM generate_series(1,3) As e CROSS JOIN generate_series(1,4) As t;
tea

{{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4}}
```

#### Ver también

[TopoElement](#), [TopoElementArray](#)

### 8.9.4 TopoElement

`TopoElement` — Converts a `topogeometry` to a `topoelement`.

#### Synopsis

`topoelement` **`TopoElement`**(`topogeometry topo`);



## Descripción

Converts a **TopoGeometry** to a **TopoElement**.

Availability: 3.4.0

## Ejemplos

Se trata de un flujo de trabajo autónomo completo

```
-- do this if you don't have a topology setup already
-- Creates topology not allowing any tolerance
SELECT TopoElement(topo)
FROM neighborhoods;
```

```
-- using as cast
SELECT topology.TopoElementArray_Agg(topo::topoelement)
FROM neighborhoods
GROUP BY city;
```

## Ver también

**TopoElementArray\_Agg**, **TopoGeometry**, **TopoElement**

## 8.10 Editores TopoGeometry

### 8.10.1 clearTopoGeom

clearTopoGeom — Borra el contenido de una topo geometry.

## Synopsis

topogeometry **clearTopoGeom**(topogeometry topogeom);

## Descripción

Borra el contenido de un **TopoGeometry** convirtiéndolo en uno vacío. Sobre todo útil en conjunción con **toTopoGeom** para substituir la forma de objetos existentes y de cualquier objeto dependiente en niveles jerárquicos más altos.

Disponibilidad: 2.1

## Ejemplos

```
-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);
```

## Ver también

**toTopoGeom**

---

### 8.10.2 TopoGeom\_addElement

TopoGeom\_addElement — Agrega un elemento a la definición de una TopoGeometry.

#### Synopsis

topogeometry **TopoGeom\_addElement**(topogeometry tg, topoelement el);

#### Descripción

Agrega un **TopoElement** a la definición de un objeto TopoGeometry. No se produce un error si el elemento ya forma parte de la definición.

Disponibilidad: 2.3

#### Ejemplos

```
-- Add edge 5 to TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_addElement(tg, '{5,2}');
```

#### Ver también

**TopoGeom\_remElement**, **CreateTopoGeom**

### 8.10.3 TopoGeom\_remElement

TopoGeom\_remElement — Quita un elemento de la definición de una TopoGeometry.

#### Synopsis

topogeometry **TopoGeom\_remElement**(topogeometry tg, topoelement el);

#### Descripción

Elimina un **TopoElement** de la definición de un objeto TopoGeometry.

Disponibilidad: 2.3

#### Ejemplos

```
-- Remove face 43 from TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_remElement(tg, '{43,3}');
```

#### Ver también

**TopoGeom\_addElement**, **CreateTopoGeom**

### 8.10.4 TopoGeom\_addTopoGeom

TopoGeom\_addTopoGeom — Adds element of a TopoGeometry to the definition of another TopoGeometry.

## Synopsis

topogeometry **TopoGeom\_addTopoGeom**(topogeometry tgt, topogeometry src);

## Descripción

Adds the elements of a **TopoGeometry** to the definition of another TopoGeometry, possibly changing its cached type (type attribute) to a collection, if needed to hold all elements in the source object.

The two TopoGeometry objects need be defined against the *\*same\** topology and, if hierarchically defined, need be composed by elements of the same child layer.

Availability: 3.2

## Ejemplos

```
-- Set an "overall" TopoGeometry value to be composed by all
-- elements of specific TopoGeometry values
UPDATE mylayer SET tg_overall = TopoGeom_addTopogeom(
 TopoGeom_addTopoGeom(
 clearTopoGeom(tg_overall),
 tg_specific1
),
 tg_specific2
);
```

## Ver también

**TopoGeom\_addElement**, **clearTopoGeom**, **CreateTopoGeom**

## 8.10.5 toTopoGeom

toTopoGeom — Agrega una forma de geometría a una geometría topográfica existente.

## Descripción

Consulte **toTopoGeom**.

## 8.11 Descriptores de Geometría Topográfica

### 8.11.1 GetTopoGeomElementArray

GetTopoGeomElementArray — Devuelve un `topoelementarray` (una matriz de `topoelements`) que contiene los elementos topológicos y el tipo de la TopoGeometry (elementos primitivos) especificados.

## Synopsis

topoelementarray **GetTopoGeomElementArray**(varchar toponame, integer layer\_id, integer tg\_id);

topoelementarray **GetTopoGeomElementArray**(topogeometry tg);

**Descripción**

Devuelve un **TopoElementArray** que contiene los elementos topológicos y el tipo de los TopoGeometry dados (elementos primitivos). Esto es similar a `GetTopoGeomElements`, excepto que devuelve los elementos como un array en lugar de un conjunto de datos.

`tg_id` es el identificador topogeometry del objeto topogeometry en la topología de la capa denotada por `layer_id` en la tabla `topology.layer`.

Availability: 1.1

**Ejemplos****Ver también**

**GetTopoGeomElements**, **TopoElementArray**

**8.11.2 GetTopoGeomElements**

`GetTopoGeomElements` — Devuelve un conjunto de objetos `topoelement` que contienen el `element_id` topológico, `element_type` de la TopoGeometry (elementos primitivos) especificados.

**Synopsis**

```
setof topoelement GetTopoGeomElements(varchar toponame, integer layer_id, integer tg_id);
setof topoelement GetTopoGeomElements(topogeometry tg);
```

**Descripción**

Returns a set of `element_id,element_type` (topoelements) corresponding to primitive topology elements **TopoElement** (1: nodes, 2: edges, 3: faces) that a given topogeometry object in `toponame` schema is composed of.

`tg_id` es el identificador topogeometry del objeto topogeometry en la topología de la capa denotada por `layer_id` en la tabla `topology.layer`.

Disponibilidad: 2.0.0

**Ejemplos****Ver también**

**GetTopoGeomElementArray**, **TopoElement**, **TopoGeom\_addElement**, **TopoGeom\_remElement**

**8.11.3 ST\_SRID**

`ST_SRID` — Returns the spatial reference identifier for a topogeometry.

**Synopsis**

```
integer ST_SRID(topogeometry tg);
```

## Descripción

Returns the spatial reference identifier for the ST\_Geometry as defined in spatial\_ref\_sys table. Section [4.5](#)



### Note

spatial\_ref\_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries.

Availability: 3.2.0



This method implements the SQL/MM specification. SQL-MM 3: 14.1.5

## Ejemplos

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)', 4326));
--result
4326
```

## Ver también

Section [4.5](#), [ST\\_SetSRID](#), [ST\\_Transform](#), [ST\\_SRID](#)

## 8.12 Salidas de Geometría Topográfica

### 8.12.1 AsGML

AsGML — Devuelve una representación GML de una geometría topográfica.

#### Synopsis

```
text AsGML(topogeometry tg);
text AsGML(topogeometry tg, text nsrefix_in);
text AsGML(topogeometry tg, regclass visitedTable);
text AsGML(topogeometry tg, regclass visitedTable, text nsrefix);
text AsGML(topogeometry tg, text nsrefix_in, integer precision, integer options);
text AsGML(topogeometry tg, text nsrefix_in, integer precision, integer options, regclass visitedTable);
text AsGML(topogeometry tg, text nsrefix_in, integer precision, integer options, regclass visitedTable, text idprefix);
text AsGML(topogeometry tg, text nsrefix_in, integer precision, integer options, regclass visitedTable, text idprefix, int gmlversion);
```

#### Descripción

Devuelve la representación GML de un topogeometry en formato de versión GML3. Si no se especifica ningún nsrefix\_in entonces se utiliza gml. Pase una cadena vacía a nsrefix para obtener un espacio de nombre no cualificado. Los parámetros de precisión (predeterminado: 15) y opciones (predeterminado 1), si se dan, se pasan sin tocar a la llamada subyacente a ST\_AsGML.

El parámetro visitedTable se utiliza para realizar un seguimiento de los elementos de nodo y arista visitados de modo que se utilicen referencias cruzadas (xlink:xref) en lugar de duplicar las definiciones. Se espera que la tabla tenga (al menos) dos campos enteros: 'element\_type' y 'element\_id'. El usuario que llama debe tener privilegios de lectura y escritura en la tabla dada. Para el mejor funcionamiento, un índice se debe definir en element\_type y element\_id, en ese orden. Dicho índice se crearía automáticamente añadiendo una restricción única a los campos. Ejemplo:

```
CREATE TABLE visited (
 element_type integer, element_id integer,
 unique(element_type, element_id)
);
```

El parámetro `idprefix`, si se da, se añadirá a los identificadores de etiquetas Edge y Node.

El parámetro `gmlver`, si se da, se pasará al ST\_AsGML subyacente. Valor predeterminado a 3.

Disponibilidad: 2.0.0

## Ejemplos

Este usa la geometría topográfica creada en [CreateTopoGeom](#)

```
SELECT topology.AsGML(topo) As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';

-- rdgml--
<gml:TopoCurve>
 <gml:directedEdge>
 <gml:Edge gml:id="E1">
 <gml:directedNode orientation="-">
 <gml:Node gml:id="N1"/>
 </gml:directedNode>
 <gml:directedNode>
 </gml:directedNode>
 <gml:curveProperty>
 <gml:Curve srsName="urn:ogc:def:crs:EPSG::3438">
 <gml:segments>
 <gml:LineStringSegment>
 <gml:posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
384833 236884 384844 236882 384866 236881 384879 236883 384954 ←
236898 385087 236932 385117 236938
385167 236938 385203 236941 385224 236946 385233 236950 385241 ←
236956 385254 236971
385260 236979 385268 236999 385273 237018 385273 237037 385271 ←
237047 385267 237057 385225 237125
385210 237144 385192 237161 385167 237192 385162 237202 385159 ←
237214 385159 237227 385162 237241
385166 237256 385196 237324 385209 237345 385234 237375 385237 ←
237383 385238 237399 385236 237407
385227 237419 385213 237430 385193 237439 385174 237451 385170 ←
237455 385169 237460 385171 237475
385181 237503 385190 237521 385200 237533 385206 237538 385213 ←
237541 385221 237542 385235 237540 385242 237541
385249 237544 385260 237555 385270 237570 385289 237584 385292 ←
237589 385291 237596 385284 237630</gml:posList>
 </gml:LineStringSegment>
 </gml:segments>
 </gml:Curve>
 </gml:curveProperty>
 </gml:Edge>
</gml:directedEdge>
</gml:TopoCurve>
>
```

Mismo ejercicio que el anterior sin espacio de nombre.

```

SELECT topology.AsGML(topo, '') As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';

-- rdgml--
<TopoCurve>
 <directedEdge>
 <Edge id="E1">
 <directedNode orientation="-">
 <Node id="N1"/>
 </directedNode>
 <directedNode
></directedNode>
 <curveProperty>
 <Curve srsName="urn:ogc:def:crs:EPSG::3438">
 <segments>
 <LineStringSegment>
 <posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
 384833 236884 384844 236882 384866 236881 384879 236883 384954 ←
 236898 385087 236932 385117 236938
 385167 236938 385203 236941 385224 236946 385233 236950 385241 ←
 236956 385254 236971
 385260 236979 385268 236999 385273 237018 385273 237037 385271 ←
 237047 385267 237057 385225 237125
 385210 237144 385192 237161 385167 237192 385162 237202 385159 ←
 237214 385159 237227 385162 237241
 385166 237256 385196 237324 385209 237345 385234 237375 385237 ←
 237383 385238 237399 385236 237407
 385227 237419 385213 237430 385193 237439 385174 237451 385170 ←
 237455 385169 237460 385171 237475
 385181 237503 385190 237521 385200 237533 385206 237538 385213 ←
 237541 385221 237542 385235 237540 385242 237541
 385249 237544 385260 237555 385270 237570 385289 237584 385292 ←
 237589 385291 237596 385284 237630</posList>
 </LineStringSegment>
 </segments>
 </Curve>
 </curveProperty>
 </Edge>
 </directedEdge>
</TopoCurve>
>

```

## Ver también

[CreateTopoGeom](#), [ST\\_CreateTopoGeo](#)

## 8.12.2 AsTopoJSON

AsTopoJSON — Devuelve la representación TopoJSON de una topogeometry.

### Synopsis

```
text AsTopoJSON(topogeometry tg, regclass edgeMapTable);
```

## Descripción

Devuelve la representación TopoJSON de un topogeometry. Si `edgeMapTable` no es nulo, se utilizará como una asignación de búsqueda/almacenamiento de identificadores de arista para los índices de arco.. Esto es para poder permitir un array compacto de "arcos" en el documento final.

Se espera que la tabla, si se da, tenga un campo "arc\_id" de tipo "serial" y un "edge\_id" de tipo entero; el código consultará la tabla para "edge\_id", por lo que se recomienda agregar un índice en ese campo.



### Note

Los índices de arco en la salida TopoJSON son de base 0, pero están en base 1 en la tabla "edgeMapTable".

Un documento completo de TopoJSON será necesario contener, además de los fragmentos devueltos por esta función, los arcos reales más algunos encabezados. Ver también la [especificación TopoJSON](#).

Disponibilidad: 2.1.0

Mejora: 2.2.1 agrega soporte para entradas puntuales.

## Ver también

[ST\\_AsGeoJSON](#)

## Ejemplos

```
CREATE TEMP TABLE edgemap(arc_id serial, edge_id int unique);

-- header
SELECT '{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects ←
": { '

-- objects
UNION ALL SELECT ''' || feature_name || '": ' || AsTopoJSON(feature, 'edgemap')
FROM features.big_parcel¶s WHERE feature_name = 'P3P4';

-- arcs
WITH edges AS (
 SELECT m.arc_id, e.geom FROM edgemap m, city_data.edge e
 WHERE e.edge_id = m.edge_id
), points AS (
 SELECT arc_id, (st_dumppoints(geom)).* FROM edges
), compare AS (
 SELECT p2.arc_id,
 CASE WHEN p1.path IS NULL THEN p2.geom
 ELSE ST_Translate(p2.geom, -ST_X(p1.geom), -ST_Y(p1.geom))
 END AS geom
 FROM points p2 LEFT OUTER JOIN points p1
 ON (p1.arc_id = p2.arc_id AND p2.path[1] = p1.path[1]+1)
 ORDER BY arc_id, p2.path
), arcdump AS (
 SELECT arc_id, (regexp_matches(ST_AsGeoJSON(geom), '\[.*\]'))[1] as t
 FROM compare
), arcs AS (
 SELECT arc_id, '[' || array_to_string(array_agg(t), ',') || ']' as a FROM arcdump
 GROUP BY arc_id
 ORDER BY arc_id
)
```



```

SELECT '}', "arcs": [' UNION ALL
SELECT array_to_string(array_agg(a), E'\n') from arcs

-- footer
UNION ALL SELECT '}]':::text as t;

-- Result:
{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects": {
"P3P4": { "type": "MultiPolygon", "arcs": [[[-1]], [[6,5,-5,-4,-3,1]]]}
}, "arcs": [
 [[25,30],[6,0],[0,10],[-14,0],[0,-10],[8,0]],
 [[35,6],[0,8]],
 [[35,6],[12,0]],
 [[47,6],[0,8]],
 [[47,14],[0,8]],
 [[35,22],[12,0]],
 [[35,14],[0,8]]
]]}

```

## 8.13 Relaciones espaciales de topología

### 8.13.1 Equals

Equals — Devuelve true si dos topogeometries están compuestas de las mismas primitivas topológicas.

#### Synopsis

boolean **Equals**(topogeometry tg1, topogeometry tg2);

#### Descripción

Devuelve verdadero si dos topogeometries se componen de las mismas primitivas de topología: caras, aristas, nodos.



#### Note

Esta función no es compatible con topogeometries que son colecciones de geometría. Tampoco puede comparar topogeometries de diferentes topologías.

Disponibilidad: 1.1.0



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

#### Ejemplos

#### Ver también

[GetTopoGeomElements](#), [ST\\_Equals](#)

### 8.13.2 Intersects

**Intersects** — Devuelve verdadero si cualquier par de primitivas de las dos topogeometries se intersectan.

#### Synopsis

```
boolean Intersects(topogeometry tg1, topogeometry tg2);
```

#### Descripción

Devuelve verdadero si cualquier par de primitivas de las dos topogeometries se intersectan.



#### Note

This function not supported for topogeometries that are geometry collections. It also can not compare topogeometries from different topologies. Also not currently supported for hierarchical topogeometries (topogeometries composed of other topogeometries).

Disponibilidad: 1.1.0



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

#### Ejemplos

#### Ver también

[ST\\_Intersects](#)

## 8.14 Importing and exporting Topologies

Once you have created topologies, and maybe associated topological layers, you might want to export them into a file-based format for backup or transfer into another database.

Using the standard dump/restore tools of PostgreSQL is problematic because topologies are composed by a set of tables (4 for primitives, an arbitrary number for layers) and records in metadata tables (topology.topology and topology.layer). Additionally, topology identifiers are not univoque across databases so that parameter of your topology will need to be changes upon restoring it.

In order to simplify export/restore of topologies a pair of executables are provided: `pgtopo_export` and `pgtopo_import`. Example usage:

```
pgtopo_export dev_db topo1 | pgtopo_import topo1 | psql staging_db
```

### 8.14.1 Using the Topology exporter

The `pgtopo_export` script takes the name of a database and a topology and outputs a dump file which can be used to import the topology (and associated layers) into a new database.

By default `pgtopo_export` writes the dump file to the standard output so that it can be piped to `pgtopo_import` or redirected to a file (refusing to write to terminal). You can optionally specify an output filename with the `-f` cmdline switch.

By default `pgtopo_export` includes a dump of all layers defined against the given topology. This may be more data than you need, or may be non-working (in case your layer tables have complex dependencies) in which case you can request skipping the layers with the `--skip-layers` switch and deal with those separately.

Invoking `pgtopo_export` with the `--help` (or `-h` for short) switch will always print short usage string.

The dump file format is a compressed tar archive of a `pgtopo_export` directory containing at least a `pgtopo_dump_version` file with format version info. As of version 1 the directory contains tab-delimited CSV files with data of the topology primitive tables (node, edge\_data, face, relation), the topology and layer records associated with it and (unless `--skip-layers` is given) a custom-format PostgreSQL dump of tables reported as being layers of the given topology.

### 8.14.2 Using the Topology importer

The `pgtopo_import` script takes a `pgtopo_export` format topology dump and a name to give to the topology to be created and outputs an SQL script reconstructing the topology and associated layers.

The generated SQL file will contain statements that create a topology with the given name, load primitive data in it, restores and registers all topology layers by properly linking all TopoGeometry values to their correct topology.

By default `pgtopo_import` reads the dump from the standard input so that it can be used in conjunction with `pgtopo_export` in a pipeline. You can optionally specify an input filename with the `-f` cmdline switch.

By default `pgtopo_import` includes in the output SQL file the code to restore all layers found in the dump.

This may be unwanted or non-working in case your target database already have tables with the same name as the ones in the dump. In that case you can request skipping the layers with the `--skip-layers` switch and deal with those separately (or later).

SQL to only load and link layers to a named topology can be generated using the `--only-layers` switch. This can be useful to load layers AFTER resolving the naming conflicts or to link layers to a different topology (say a spatially-simplified version of the starting topology).

## Chapter 9

# Gestión, Consulta y Aplicaciones de Datos Raster

### 9.1 Cargando y Creando Rasters

En la mayoría de casos, crearás rasters PostGIS cargando un fichero raster utilizando el paquete de carga raster `raster2pgsql`.

#### 9.1.1 Utilizar el paquete `raster2pgsql` para cargar rasters

The `raster2pgsql` is a raster loader executable that loads GDAL supported raster formats into SQL suitable for loading into a PostGIS raster table. It is capable of loading folders of raster files as well as creating overviews of rasters.

Since the `raster2pgsql` is compiled as part of PostGIS most often (unless you compile your own GDAL library), the raster types supported by the executable will be the same as those compiled in the GDAL dependency library. To get a list of raster types your particular `raster2pgsql` supports use the `-G` switch.

**Note**

Cuando creamos previsualizaciones de un factor específico de un conjunto de rasters que están alineados, es posible que las previsualizaciones no estén alineadas. Visita <http://trac.osgeo.org/postgis/ticket/1764> para un ejemplo donde las previsualizaciones no se alinean.

##### 9.1.1.1 Example Usage

Un ejemplo de sesión utilizando el cargador para crear un fichero de entrada y cargarlo cortado en teselas de 100x100 debería parecerse a:

```
-s use srid 4326
-I create spatial index
-C use standard raster constraints
-M vacuum analyze after load
*.tif load all these files
-F include a filename column in the raster table
-t tile the output 100x100
public.demelevation load into this table
raster2pgsql -s 4326 -I -C -M -F -t 100x100 *.tif public.demelevation
> elev.sql

-d connect to this database
-f read this file after connecting
psql -d gisdb -f elev.sql
```

**Note**

If you do not specify the schema as part of the target table name, the table will be created in the default schema of the database or user you are connecting with.

Se puede hacer una conversión y carga en un solo paso con el carácter "|" en sistemas UNIX:

```
raster2pgsql -s 4326 -I -C -M *.tif -F -t 100x100 public.demelevation | psql -d gisdb
```

Carga las teselas planas métricas aéreas de los Raster del estado Massachusetts en un esquema denominado `aerial` y crear una vista completa, y previsualizaciones de niveles 2 y 4, utiliza el modo de copia para insertar (sin archivo intermedio sólo directamente a db), y `-e` no fuerces todo en una transacción (bueno si quieres ver datos en tablas de inmediato sin tener que esperar). Divide los raster en teselas de 128x128 píxeles y aplica las restricciones de raster. Utiliza el modo copia en lugar de insertar en la tabla. `(-F)` Incluye un campo llamado nombre de archivo para contener el nombre del archivo de las teselas de donde proceden los cortes.

```
raster2pgsql -I -C -e -Y -F -s 26986 -t 128x128 -l 2,4 bostonaerials2008/*.jpg aerials. ↵
 boston | psql -U postgres -d gisdb -h localhost -p 5432
```

```
--get a list of raster types supported:
raster2pgsql -G
```

El comando `-G` extrae una lista similar a esta:

```
Available GDAL raster formats:
Virtual Raster
GeoTIFF
National Imagery Transmission Format
Raster Product Format TOC format
ECRG TOC format
Erdas Imagine Images (.img)
CEOS SAR Image
CEOS Image
...
Arc/Info Export E00 GRID
ZMap Plus Grid
NOAA NGS Geoid Height Grids
```

### 9.1.1.2 raster2pgsql options

**-?** Muestra la pantalla de ayuda. También se mostrará la ayuda si no incluyes ningún argumento.

**-G** Imprime los formatos raster soportados.

**(claldp)** Estas opciones son exclusivas de forma mutua:

- c** Crea una nueva tabla y la rellena con el raster(s). *Esta es la opción por defecto.*
- a** Añade el/los raster/s a una tabla existente.
- d** Borra la tabla, crea una nueva y la rellena con el/los raster(s)
- p** Modo preparación, solo crea la tabla.

**Procesamiento Raster: Añade condiciones para registrar de forma limpia en el catalogo raster**

- C** Añade restricciones raster `--srid`, tamaño del pixel, etc. para asegurar que el raster es registrado de forma correcta en la vista `raster_columns`.
- x** Desactiva la opción de restricción de máxima extensión. Solo se aplica si la opción `-C` esta en uso.

- r Establezca las restricciones (espacialmente única y tesela de cobertura) para el bloqueo regular. Sólo se aplica si la bandera -C también está en uso.

### Procesado Raster: Parámetros opcionales utilizados para manipular la entrada de datos raster

- s <SRID> Asigna un SRID específico al raster de salida. Si no se especifica o es igual a 0, se comprueban los metadatos del raster para determinar un SRID apropiado.
- b **BAND** Índice (en base 1) de la banda para extraer del raster. Para índices de más de una banda, sepáralo con comas(.). Si no se especifica, se extraerán todas las bandas del raster.
- t **TILE\_SIZE** Cortar el ráster en teselas para ser insertadas una por una en registros de la tabla. `TILE_SIZE` se expresa como `ANCHOxALTO` o establecer el valor "auto" para permitir que se cargue a la computadora en un tamaño de tesela apropiado utilizando el primer ráster y aplicarlo a todos los rústers.
- P Pad right-most and bottom-most tiles to guarantee that all tiles have the same width and height.
- R, --register Registra el raster como un fichero de sistema (out-db) raster.  
Solo los metadatos del raster y el camino de acceso al raster se almacenan en la base de datos (no los píxeles).
- l **OVERVIEW\_FACTOR** Crear una previsualización del ráster. Para más de un factor, separar con coma(.). El nombre de la tabla de la previsualización sigue el patrón `o_factor de previsualización_tabla`, donde `factor previsualización` es un marcador de posición para el factor de previsualización numérica y `tabla` se reemplaza con el nombre de la tabla base. La previsualización creada es almacenada en la base de datos y no se afecta por -R. Tenga en cuenta que su archivo sql generado contendrá ambas, la tabla principal y las tablas de previsualización.
- N **NODATA** Valor NODATA para utilizar en bandas con valores NODATA.

### Parámetros opcionales para manipular objetos de la base de datos

- f **COLUMN** Especifica el nombre de la columna raster de destino, por defecto es 'rast'
- F Añade una columna con el nombre del fichero
- n **COLUMN** Specify the name of the filename column. Implies -F.
- q Wrap PostgreSQL identifiers in quotes.
- I Crea un índice GiST de la columna raster.
- M Ejecuta Vacuum analyze en la tabla raster.
- k Keeps empty tiles and skips NODATA value checks for each raster band. Note you save time in checking, but could end up with far more junk rows in your database and those junk rows are not marked as empty tiles.
- T **tablespace** Especifica el "tablespace" de la nueva tabla. Observa que los índices (incluyendo el de clave primaria) seguirá utilizando en "tablespace" a menos que se utilice también la opción -X.
- X **tablespace** Especifica el "tablespace" para el nuevo índice de la tabla. Esto se aplica a los índices de claves primarias y índices espaciales si la opción -I se está usando.
- Y **max\_rows\_per\_copy=50** Use copy statements instead of insert statements. Optionally specify `max_rows_per_copy`; default 50 when not specified.
- e Ejecuta cada comando de forma individual, no utiliza transacciones.
- E **ENDIAN** Controla el formato en el que se almacenan los datos de más de un byte (endianness) de la salida binaria generada del raster; especifica 0 para XDR y 1 para NDR (por defecto); solo las salidas NDR están soportadas actualmente.
- V **versión** Especifica la versión del formato de salida. Por defecto es 0. Solo 0 está soportado actualmente.

## 9.1.2 Crear rastres utilizando las funciones raster de PostGIS

En muchas ocasiones, querrás crear tablas raster en la base de datos. Existen una gran cantidad de funciones para hacerlo. Los pasos generales a seguir.

1. Crear una tabla con una columna raster para almacenar los nuevos registros raster se puede hacer de la siguiente manera:

```
CREATE TABLE myrasters(rid serial primary key, rast raster);
```

- Existen muchas funciones de ayuda. Si no estas creando rasters con derivados de otro raster, entonces deberías comenzar con: **ST\_MakeEmptyRaster**, seguido de **ST\_AddBand**

También puedes crear rasters a partir de geometrias. Para conseguir esto deberás utilizar **ST\_AsRaster** quizás acompañado de otras funciones como **ST\_Union** o **ST\_MapAlgebraFct** o cualquier otra de la familia de funciones de álgebra de mapas.

Incluso hay muchas más opciones para crear nuevas tablas raster a partir de las tablas existentes. Por ejemplo, puede crear una tabla raster en una proyección diferente de una existente utilizando **ST\_Transform**

- Una vez que hayas terminado de llenar la tabla, tendrás que crear un índice espacial en la columna raster con algo similar a:

```
CREATE INDEX myrasters_rast_st_convexhull_idx ON myrasters USING gist(ST_ConvexHull(↵
rast));
```

Observa que utilizamos **ST\_ConvexHull** ya que muchas de las operaciones raster se basan en la envolvente convexa del raster.



#### Note

En versiones anteriores a PostGIS 2.0 los raster se basaban en la envolvente y no en la envolvente convexa. Para que los índices espaciales funcionen correctamente necesitaras borrarlos y reemplazarlos por los índices basados en la envolvente convexa.

- Aplica las restricciones raster con **AddRasterConstraints**

### 9.1.3 Using "out db" cloud rasters

The `raster2pgsql` tool uses GDAL to access raster data, and can take advantage of a key GDAL feature: the ability to read from rasters that are **stored remotely** in cloud "object stores" (e.g. AWS S3, Google Cloud Storage).

Efficient use of cloud stored rasters requires the use of a "cloud optimized" format. The most well-known and widely used is the **"cloud optimized GeoTIFF"** format. Using a non-cloud format, like a JPEG, or an un-tiled TIFF will result in very poor performance, as the system will have to download the entire raster each time it needs to access a subset.

First, load your raster into the cloud storage of your choice. Once it is loaded, you will have a URI to access it with, either an "http" URI, or sometimes a URI specific to the service. (e.g., "s3://bucket/object"). To access non-public buckets, you will need to supply GDAL config options to authenticate your connection. Note that this command is *reading* from the cloud raster and *writing* to the database.

```
AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxxx \
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \
raster2pgsql \
-s 990000 \
-t 256x256 \
-I \
-R \
/vsis3/your.bucket.com/your_file.tif \
your_table \
| psql your_db
```

Once the table is loaded, you need to give the database permission to read from remote rasters, by setting two permissions, **postgis.enable\_outdb\_rasters** and **postgis.gdal\_enabled\_drivers**.

```
SET postgis.enable_outdb_rasters = true;
SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

To make the changes sticky, set them directly on your database. You will need to re-connect to experience the new settings.

```
ALTER DATABASE your_db SET postgis.enable_outdb_rasters = true;
ALTER DATABASE your_db SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

For non-public rasters, you may have to provide access keys to read from the cloud rasters. The same keys you used to write the `raster2pgsql` call can be set for use inside the database, with the `postgis.gdal_config_options` configuration. Note that multiple options can be set by space-separating the `key=value` pairs.

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
```

Once you have the data loaded and permissions set you can interact with the raster table like any other raster table, using the same functions. The database will handle all the mechanics of connecting to the cloud data when it needs to read pixel data.

## 9.2 Catalogos raster

Existen dos vistas de catalogo raster que vienen en el paquete PostGIS. Ambas vistas utilizan información de las restricciones de las tablas raster. Como resultado, las vistas de catálogo tienen siempre consistencia con los datos raster de las tablas mientras que las restricciones son reforzadas.

1. La vista `raster_columns` cataloga todas las columnas raster de todas las tablas de la base de datos.
2. La vista `raster_overviews` cataloga todas las columnas raster de las tablas de la base de datos que sirven como previsualizaciones de tablas de grano más fino. Las tablas de este tipo se generan cuando utilizas la opción `-l` durante la carga.

### 9.2.1 Catalogo de columnas raster

El catálogo `raster_columns` es un catálogo de todas las columnas de las tablas raster en la base de datos que son de tipo raster. Es una vista que utiliza las restricciones de las tablas por lo que la información es siempre consistente, incluso si se restaura una tabla raster de una copia de seguridad de otra base de datos. Existen las siguientes columnas en el catálogo `raster_columns`.

Si has creado tus tablas sin el cargador o has olvidado especificar la variable `-C` del comando de carga durante la carga, puedes hacer cumplir las restricciones por defecto utilizando `AddRasterConstraints`, de este modo el catálogo `raster_columns` guardará la información mas común de tus teselas raster.

- `r_table_catalog` Contienen la tabla de la base de datos. Esto siempre leerá la base de datos actual.
- `r_table_schema` Esquema al que pertenece la tabla.
- `r_table_name` tabla raster
- `r_raster_column` Columna de la tabla `r_table_name` que es de tipo raster. No hay nada en PostGIS que impida tener múltiples columnas raster por tabla así que es posible tener varias veces la misma tabla raster en la lista con diferentes columnas cada vez.
- `srid` El identificador del sistema de referencia espacial del raster. Debe ser una de las entradas de la tabla [Section 4.5](#).
- `scale_x` Escala entre las coordenadas espaciales geométricas y el pixel. Esto esta disponible únicamente si todas las teselas de la columna raster tienen el mismo valor `scale_x` y se aplica la restricción. Para mas detalles visita [ST\\_ScaleX](#).
- `scale_y` Escala entre las coordenadas espaciales geométricas y el pixel. Esto esta disponible únicamente si todas las teselas de la columna raster tienen el mismo valor `scale_y` y se aplica la restricción `scale_y`. Para mas detalles visita [ST\\_ScaleY](#).
- `blocksize_x` Es el ancho (numero de pixeles en horizontal) de cada tesela raster. Para mas detalles visita [ST\\_Width](#).
- `blocksize_y` Es el ancho (numero de pixeles en vertical hacia abajo) de cada tesela raster. Para mas detalles visita [ST\\_Height](#).
- `same_alignment` Valor booleano que valdrá "True" si todas las teselas raster tienen el mismo alineamiento. Visita [ST\\_SameAlignment](#) para más información.



- `regular_blocking` Si la columna del ráster tiene las limitaciones de espacio único y de cobertura de tesela, el valor es `TRUE`. De lo contrario, será `FALSE`.
- `num_bands` Numero de bandas por tesela del conjunto de rasters. Es la misma información que la devuelta por `ST_NumBands`
- `pixel_types` Un array definiendo el tipo de pixel de cada banda. Tendrás el mismo numero de elementos en este array que el numero de bandas. Los `pixel_types` son uno de los definidos en `ST_BandPixelType`.
- `nodata_values` Un array de números de doble precisión que define el valor `nodata_value` de cada banda. En este array deberás tener el mismo numero de elementos que el numero de bandas. Este numero define el valor de los pixeles de cada banda que deben ignorarse para la mayoría de operaciones. Esta información es similar a la proporcionada por `ST_BandNoDataValue`.
- `out_db` Una colección de banderas booleanas indican si los datos de las bandas del ráster se mantienen fuera de la base de datos. Se tendrá el mismo número de elementos en esta colección como se tiene número de bandas.
- `extent` Esta es la extensión de todas las columnas raster en tu conjunto raster. Si planeas cargar mas datos que cambiarán la extensión del conjunto, deberás ejecutar la función `DropRasterConstraints` antes de la carga y después de la carga restablecer las restricciones con `AddRasterConstraints`.
- `spatial_index` Un boolean es verdadero si la columna del ráster tiene un índice espacial.

### 9.2.2 Previsualizaciones raster

`raster_overviews` Los catálogos de información acerca de las columnas de tablas ráster utilizadas para las previsualizaciones e información adicional de ellos que son útiles para conocer cuando utilizar vistas generales. Las tablas de previsualización Overview tables están catalogados tanto en `raster_columns` y `raster_overviews` porque son rásters en su propio derecho pero también sirven a un propósito especial adicional de ser una caricatura de resolución baja o de una tabla de resolución alta. Estos se generan a lo largo de la tabla ráster principal cuando se utiliza el `-1` interruptor en la carga del ráster o se puede generar manualmente utilizando `AddOverviewConstraints`.

Las tablas de previsualización contienen las mismas restricciones que cualquier tabla raster además de restricciones adicionales específicas a las previsualizaciones.



#### Note

La información de la tabla `raster_overviews` no duplica la información de `raster_columns`. Si necesitas información sobre una tabla de previsualizaciones pobremente en `raster_columns` puedes unir la tabla `raster_overviews` a `raster_columns` para obtener toda la información que necesitas.

Las dos principales razones de crear previsualizaciones son:

1. Tener una representación de baja resolución de las tablas principales para tener una respuesta rápida en operaciones de zoom-out.
2. Los cálculos son generalmente más rápidos en las previsualizaciones que en las imágenes de mayor resolución porque hay menos registros y cada pixel cubre más territorio. Aunque los cálculos no son tan precisos como en las tablas de alta resolución de las que provienen, pueden ser suficientes en muchos cálculos empíricos.

El catálogo `raster_overviews` contiene las siguientes columnas de información.

- `o_table_catalog` La base de datos a la cual pertenece la tabla de previsualizaciones. Esto siempre leerá la base de datos actual.
- `o_table_schema` El esquema de la base de datos al cual pertenece la tabla de previsualizaciones.
- `o_table_name` El nombre de la tala de previsualizaciones.
- `o_raster_column` La columna raster de la tabla de previsualizaciones.

- `r_table_catalog` La base de datos de la tabla raster para la cual esta previsualización sirve. Esto siempre va a leer la base de datos actual.
- `r_table_schema` El esquema de la base de datos de la tabla ráster al cual pertenecen estas previsualizaciones.
- `r_table_name` tabla raster para la cual sirven las previsualizaciones.
- `r_raster_column` la columna raster para la cual sirven estas previsualizaciones.
- `overview_factor` - este es el nivel de pirámide de la tabla de previsualizaciones . Cuanto más alto sea el número, más baja es la resolución de la tabla. Si se le da una carpeta de imágenes al comando `raster2pgsql`, se calcularán previsualizaciones de cada archivo de imagen y se cargarán por separado. El Nivel 1 supone siempre el archivo original. Nivel 2 tendrá cada tesela representada por 4 de la original. Por ejemplo, si tienes una carpeta de archivos de imagen de 5000x5000 pixeles que decidiste dividir en imágenes de 125x125 , para presentar cada imagen tu tabla base tendrá  $(5000*5000)/(125*125)=1.600$  registros , tu tabla (`l=2`) `o_2` tendrá un tope de  $(1600/Potencia(2,2))=400$  filas , tu (`l=3`) `o_3` tendrá un tope de  $(1600/Potencia(2,3)) = 200$  filas. Si los píxeles no son divisibles por el tamaño de tus teselas , obtendrás algunas de relleno (teselas no completamente llenas) . Ten en cuenta que cada tesela de previsualización generada por el comando `raster2pgsql` tiene el mismo número de pixeles que la tesela de origen , pero es de menor resolución que cada pixel de la que representa ( $Potencia(2,factor\_de\_previsualizacion)$  pixeles del original) .

## 9.3 Contruyendo aplicaciones personalizadas con PostGIS Raster

The fact that PostGIS raster provides you with SQL functions to render rasters in known image formats gives you a lot of options for rendering them. For example you can use OpenOffice / LibreOffice for rendering as demonstrated in [Rendering PostGIS Raster graphics with LibreOffice Base Reports](#). In addition you can use a wide variety of languages as demonstrated in this section.

### 9.3.1 Ejemplo de salida utilizando ST\_AsPNG junto con otras opciones raster en PHP

En esta sección, mostraremos como utilizar el driver PHP PostgreSQL y la familia de funciones `ST_AsGDALRaster` para extraer las nadas 1,2,3 de un raster a una consulta PHP que puede incluirse como una marca html img src.

La consulta de ejemplo muestra cómo combinar un montón de funciones de mapa de bits juntos para obtener todas las teselas que se cruzan con un cuadro delimitador en wgs84 en particular y luego unimos las teselas que intersectan con `ST_Union` devolviendo todas las bandas, transformadas al sistema de proyección específico del usuario con `ST_Transform`, y luego enviamos el resultado como un png con `ST_AsPNG`.

Se podría llamar a la continuación utilizando

```
http://mywebserver/test_raster.php?srid=2249
```

para obtener la imagen raster en pies del estado de Massachusetts.

```
<?php
/** contents of test_raster.php */
$conn_str = 'dbname=mydb host=localhost port=5432 user=myuser password=mypwd';
$dbconn = pg_connect($conn_str);
header('Content-Type: image/png');
/**If a particular projection was requested use it otherwise use mass state plane meters ↵
**/
if (!empty($_REQUEST['srid']) && is_numeric($_REQUEST['srid'])){
 $input_srid = intval($_REQUEST['srid']);
}
else { $input_srid = 26986; }
/** The set bytea_output may be needed for PostgreSQL 9.0+, but not for 8.4 **/
$sql = "set bytea_output='escape';
SELECT ST_AsPNG(ST_Transform(
 ST_AddBand(ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast ↵
 ,3]))
```

```

 , $input_srid)) As new_rast
FROM aerials.boston
WHERE
 ST_Intersects(rast, ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, -71.1210, 42.218, 4326), 26986));
$result = pg_query($sql);
$row = pg_fetch_row($result);
pg_free_result($result);
if ($row === false) return;
echo pg_unescape_bytea($row[0]);
?>

```

### 9.3.2 Ejemplo de salida utilizando ST\_AsPNG junto con otras opciones raster en ASP.NET C#

En esta sección, mostraremos como utilizar el driver Npgsql PostgreSQL .NET y la familia de funciones **ST\_AsGDALRaster** para extraer las bandas 1,2,3 de un raster a una consulta PHP que puede incluirse como una marca html `img src`.

Necesitarás el driver `npgsql` .NET PostgreSQL para este ejercicio que puedes obtener en <http://npgsql.projects.postgresql.org/> en su ultima versión. Simplemente descarga la última versión y copialo en tu carpeta bin de ASP.NET y ya estarás listo para seguir.

La consulta de ejemplo muestra cómo combinar un montón de funciones de mapa de bits juntos para obtener todas las teselas que se cruzan con un cuadro delimitador en wgs84 en particular y luego unimos las teselas que intersectan con **ST\_Union** devolviendo todas las bandas, transformadas al sistema de proyección específico del usuario con **ST\_Transform**, y luego enviamos el resultado como un png con **ST\_AsPNG**.

Este ejemplo es el mismo que el ejemplo Section 9.3.1 salvo que este esta implementado en C#.

Puedes llamar a este método utilizando

```
http://mywebserver/TestRaster.ashx?srid=2249
```

para obtener la imagen raster en coordenadas planas en pies del estado de Massachusetts.

```

-- web.config connection string section --
<connectionStrings>
 <add name="DSN"
 connectionString="server=localhost;database=mydb;Port=5432;User Id=myuser;password=
 mypwd"/>
</connectionStrings>
>

```

```

// Code for TestRaster.ashx
<%@ WebHandler Language="C#" Class="TestRaster" %>
using System;
using System.Data;
using System.Web;
using Npgsql;

public class TestRaster : IHttpHandler
{
 public void ProcessRequest(HttpContext context)
 {
 context.Response.ContentType = "image/png";
 context.Response.BinaryWrite(GetResults(context));
 }

 public bool IsReusable {
 get { return false; }
 }
}

```

```

public byte[] GetResults(HttpContext context)
{
 byte[] result = null;
 NpgsqlCommand command;
 string sql = null;
 int input_srid = 26986;
 try {
 using (NpgsqlConnection conn = new NpgsqlConnection(System. ↵
 Configuration.ConfigurationManager.ConnectionStrings["DSN"]. ↵
 ConnectionString)) {
 conn.Open();

 if (context.Request["srid"] != null)
 {
 input_srid = Convert.ToInt32(context.Request["srid"]);
 }
 sql = @"SELECT ST_AsPNG(
 ST_Transform(
 ST_AddBand(
 ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast,3)] ↵
 ,:input_srid)) As new_rast
 FROM aerials.boston
 WHERE
 ST_Intersects(rast,
 ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, ↵
 -71.1210, 42.218,4326),26986))");
 command = new NpgsqlCommand(sql, conn);
 command.Parameters.Add(new NpgsqlParameter("input_srid", input_srid));

 result = (byte[]) command.ExecuteScalar();
 conn.Close();
 }
 }
 catch (Exception ex)
 {
 result = null;
 context.Response.Write(ex.Message.Trim());
 }
 return result;
}

```

### 9.3.3 Aplicación de consola Java que extrae un raster como un fichero de imagen

Esta es una aplicación simple de la consola java que toma una consulta y devuelve una imagen y la extrae a un fichero especificado.

Puedes descargar el último driver PostgreSQL JDBC desde <http://jdbc.postgresql.org/download.html>

Puedes compilar el siguiente código utilizando un comando similar a este:

```

set env CLASSPATH\postgresql-9.0-801.jdbc4.jar
javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class

```

Y llamarlo desde la línea de comandos de forma similar a:

```

java -jar SaveQueryImage.jar "SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10, ' ↵
quad_segs=2'),150, 150, '8BUI',100));" "test.png"

```

```
-- Manifest.txt --
```

```
Class-Path: postgresql-9.0-801.jdbc4.jar
```

```
Main-Class: SaveQueryImage
```

```
// Code for SaveQueryImage.java
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.*;

public class SaveQueryImage {
 public static void main(String[] argv) {
 System.out.println("Checking if Driver is registered with DriverManager.");

 try {
 //java.sql.DriverManager.registerDriver (new org.postgresql.Driver());
 Class.forName("org.postgresql.Driver");
 }
 catch (ClassNotFoundException cnfe) {
 System.out.println("Couldn't find the driver!");
 cnfe.printStackTrace();
 System.exit(1);
 }

 Connection conn = null;

 try {
 conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/mydb","myuser ←
 ", "mypwd");
 conn.setAutoCommit(false);

 PreparedStatement sGetImg = conn.prepareStatement(argv[0]);

 ResultSet rs = sGetImg.executeQuery();

 FileOutputStream fout;
 try
 {
 rs.next();
 /** Output to file name requested by user */
 fout = new FileOutputStream(new File(argv[1]));
 fout.write(rs.getBytes(1));
 fout.close();
 }
 catch(Exception e)
 {
 System.out.println("Can't create file");
 e.printStackTrace();
 }

 rs.close();
 sGetImg.close();
 conn.close();
 }
 catch (SQLException se) {
 System.out.println("Couldn't connect: print out a stack trace and exit.");
 se.printStackTrace();
 System.exit(1);
 }
 }
}
```

```
}
```

### 9.3.4 Utilizar PLPython para extraer imágenes vía SQL

Este es una función de almacenamiento plpython que crea un archivo en el directorio del servidor por cada registro. Requiere que tenga instalado plpython. Deberá trabajar bien con ambos plpythonu y plpython3u.

```
CREATE OR REPLACE FUNCTION write_file (param_bytes bytea, param_filepath text)
RETURNS text
AS $$
f = open(param_filepath, 'wb+')
f.write(param_bytes)
return param_filepath
$$ LANGUAGE plpythonu;
```

```
--write out 5 images to the PostgreSQL server in varying sizes
-- note the postgresql daemon account needs to have write access to folder
-- this echos back the file names created;
SELECT write_file(ST_AsPNG(
 ST_AsRaster(ST_Buffer(ST_Point(1,5),j*5, 'quad_segs=2'),150*j, 150*j, '8BUI',100)),
 'C:/temp/slices'|| j || '.png')
FROM generate_series(1,5) As j;

 write_file

C:/temp/slices1.png
C:/temp/slices2.png
C:/temp/slices3.png
C:/temp/slices4.png
C:/temp/slices5.png
```

### 9.3.5 Extraer un raster con PSQL

Sadly PSQL doesn't have easy to use built-in functionality for outputting binaries. This is a bit of a hack that piggy backs on PostgreSQL somewhat legacy large object support. To use first launch your psql commandline connected to your database.

A diferencia del enfoque de python, este, crea el fichero en tu equipo local.

```
SELECT oid, lowrite(lo_open(oid, 131072), png) As num_bytes
FROM
(VALUES (lo_create(0),
 ST_AsPNG((SELECT rast FROM aeriels.boston WHERE rid=1))
)) As v(oid,png);
-- you'll get an output something like --
 oid | num_bytes
-----+-----
2630819 | 74860

-- next note the oid and do this replacing the c:/test.png to file path location
-- on your local computer
\lo_export 2630819 'C:/temp/aerial_samp.png'

-- this deletes the file from large object storage on db
SELECT lo_unlink(2630819);
```



## 10.1 Raster Support Data types

### 10.1.1 geomval

**geomval** — A spatial datatype with two fields - **geom** (holding a geometry object) and **val** (holding a double precision pixel value from a raster band).

#### Description

**geomval** is a compound data type consisting of a geometry object referenced by the **.geom** field and **val**, a double precision value that represents the pixel value at a particular geometric location in a raster band. It is used by the **ST\_DumpAsPolygon** and Raster intersection family of functions as an output type to explode a raster band into geometry polygons.

#### See Also

Section [12.6](#)

### 10.1.2 addbandarg

**addbandarg** — A composite type used as input into the **ST\_AddBand** function defining the attributes and initial value of the new band.

#### Description

A composite type used as input into the **ST\_AddBand** function defining the attributes and initial value of the new band.

**index integer** 1-based value indicating the position where the new band will be added amongst the raster's bands. If NULL, the new band will be added at the end of the raster's bands.

**pixeltype text** Pixel type of the new band. One of defined pixel types as described in [ST\\_BandPixelType](#).

**initialvalue double precision** Initial value that all pixels of new band will be set to.

**nodataval double precision** NODATA value of the new band. If NULL, the new band will not have a NODATA value assigned.

#### See Also

[ST\\_AddBand](#)

### 10.1.3 rastbandarg

**rastbandarg** — A composite type for use when needing to express a raster and a band index of that raster.

#### Description

A composite type for use when needing to express a raster and a band index of that raster.

**rast raster** The raster in question/

**nband integer** 1-based value indicating the band of raster



See Also

[ST\\_MapAlgebra \(callback function version\)](#)

10.1.4 raster

raster — raster spatial data type.

Description

raster is a spatial data type used to represent raster data such as those imported from JPEGs, TIFFs, PNGs, digital elevation models. Each raster has 1 or more bands each having a set of pixel values. Rasters can be georeferenced.



**Note** Requires PostGIS be compiled with GDAL support. Currently rasters can be implicitly converted to geometry type, but the conversion returns the [ST\\_ConvexHull](#) of the raster. This auto casting may be removed in the near future so don't rely on it.

Casting Behavior

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

| Cast To  | Behavior  |
|----------|-----------|
| geometry | automatic |

See Also

Chapter [10](#)

10.1.5 reclassarg

reclassarg — A composite type used as input into the ST\_Reclass function defining the behavior of reclassification.

Description

A composite type used as input into the ST\_Reclass function defining the behavior of reclassification.

**nband integer** The band number of band to reclassify.

**reclassexpr text** range expression consisting of comma delimited range:map\_range mappings. : to define mapping that defines how to map old band values to new band values. ( means >, ) means less than, ] < or equal, [ means > or equal

1. [a-b] = a <= x <= b

2. (a-b) = a < x <= b

3. [a-b) = a <= x < b

4. (a-b) = a < x < b

( notation is optional so a-b means the same as (a-b)

**pixeltype text** One of defined pixel types as described in [ST\\_BandPixelType](#)

**nodataval double precision** Value to treat as no data. For image outputs that support transparency, these will be blank.

**Example: Reclassify band 2 as an 8BUI where 255 is nodata value**

```
SELECT ROW(2, '0-100:1-10, 101-500:11-150,501 - 10000: 151-254', '8BUI', 255)::reclassarg;
```

**Example: Reclassify band 1 as an 1BB and no nodata value defined**

```
SELECT ROW(1, '0-100]:0, (100-255:1', '1BB', NULL)::reclassarg;
```

**See Also**

[ST\\_Reclass](#)

**10.1.6 summarystats**

**summarystats** — A composite type returned by the [ST\\_SummaryStats](#) and [ST\\_SummaryStatsAgg](#) functions.

**Description**

A composite type returned by the [ST\\_SummaryStats](#) and [ST\\_SummaryStatsAgg](#) functions.

**count integer** Number of pixels counted for the summary statistics.

**sum double precision** Sum of all counted pixel values.

**mean double precision** Arithmetic mean of all counted pixel values.

**stddev double precision** Standard deviation of all counted pixel values.

**min double precision** Minimum value of counted pixel values.

**max double precision** Maximum value of counted pixel values.

**See Also**

[ST\\_SummaryStats](#), [ST\\_SummaryStatsAgg](#)

**10.1.7 unionarg**

**unionarg** — A composite type used as input into the [ST\\_Union](#) function defining the bands to be processed and behavior of the UNION operation.

**Description**

A composite type used as input into the [ST\\_Union](#) function defining the bands to be processed and behavior of the UNION operation.

**nband integer** 1-based value indicating the band of each input raster to be processed.

**uniontype text** Type of UNION operation. One of defined types as described in [ST\\_Union](#).

**See Also**

[ST\\_Union](#)

## 10.2 Raster Management

### 10.2.1 AddRasterConstraints

**AddRasterConstraints** — Adds raster constraints to a loaded raster table for a specific column that constrains spatial ref, scaling, blocksize, alignment, bands, band type and a flag to denote if raster column is regularly blocked. The table must be loaded with data for the constraints to be inferred. Returns true if the constraint setting was accomplished and issues a notice otherwise.

#### Synopsis

```
boolean AddRasterConstraints(name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true,
boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=false, boolean
num_bands=true, boolean pixel_types=true, boolean nodata_values=true, boolean out_db=true, boolean extent=true);
boolean AddRasterConstraints(name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean AddRasterConstraints(name rastschema, name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean AddRasterConstraints(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true,
boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=false,
boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true, boolean out_db=true, boolean extent=true);
```

#### Description

Generates constraints on a raster column that are used to display information in the `raster_columns` raster catalog. The `rastschema` is the name of the table schema the table resides in. The `srid` must be an integer value reference to an entry in the `SPATIAL_REF_SYS` table.

`raster2pgsql` loader uses this function to register raster tables

Valid constraint names to pass in: refer to Section 9.2.1 for more details.

- `blocksize` sets both X and Y blocksize
- `blocksize_x` sets X tile (width in pixels of each tile)
- `blocksize_y` sets Y tile (height in pixels of each tile)
- `extent` computes extent of whole table and applies constraint all rasters must be within that extent
- `num_bands` number of bands
- `pixel_types` reads array of pixel types for each band ensure all band n have same pixel type
- `regular_blocking` sets spatially unique (no two rasters can be spatially the same) and coverage tile (raster is aligned to a coverage) constraints
- `same_alignment` ensures they all have same alignment meaning any two tiles you compare will return true for. Refer to [ST\\_SameAlignment](#).
- `srid` ensures all have same srid
- More -- any listed as inputs into the above functions



#### Note

This function infers the constraints from the data already present in the table. As such for it to work, you must create the raster column first and then load it with data.



### Note

If you need to load more data in your tables after you have already applied constraints, you may want to run the `DropRasterConstraints` if the extent of your data has changed.

Availability: 2.0.0

**Examples: Apply all possible constraints on column based on data**

```
CREATE TABLE myrasters(rid SERIAL primary key, rast raster);
INSERT INTO myrasters(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, '8BSI':: ←
 text, -129, NULL);

SELECT AddRasterConstraints('myrasters'::name, 'rast'::name);

-- verify if registered correctly in the raster_columns view --
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types, ←
 nodata_values
FROM raster_columns
WHERE r_table_name = 'myrasters';

srid | scale_x | scale_y | blocksize_x | blocksize_y | num_bands | pixel_types | ←
nodata_values
-----+-----+-----+-----+-----+-----+-----+-----+
4326 | 2 | 2 | 1000 | 1000 | 1 | {8BSI} | {0}
```

### Examples: Apply single constraint

```
CREATE TABLE public.myrasters2(rid SERIAL primary key, rast raster);
INSERT INTO myrasters2(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, '8BSI':: ↵
 text, -129, NULL);

SELECT AddRasterConstraints('public'::name, 'myrasters2'::name, 'rast'::name,' ↵
 regular_blocking', 'blocksize');
-- get notice--
NOTICE: Adding regular blocking constraint
NOTICE: Adding blocksize-X constraint
NOTICE: Adding blocksize-Y constraint
```

## See Also

Section 9.2.1, ST\_AddBand, ST\_MakeEmptyRaster, DropRasterConstraints, ST\_BandPixelType, ST\_SRID

### 10.2.2 DropRasterConstraints

**DropRasterConstraints** — Drops PostGIS raster constraints that refer to a raster table column. Useful if you need to reload data or update your raster column data.

Synopsis

boolean **DropRasterConstraints**(name rasttable, name rastcolumn, boolean srid, boolean scale\_x, boolean scale\_y, boolean blocksize\_x, boolean blocksize\_y, boolean same\_alignment, boolean regular\_blocking, boolean num\_bands=true, boolean pixel\_types=true, boolean nodata\_values=true, boolean out\_db=true , boolean extent=true);  
boolean **DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale\_x=true, boolean scale\_y=true, boolean blocksize\_x=true, boolean blocksize\_y=true, boolean same\_alignment=true, boolean regular\_blocking=false, boolean num\_bands=true, boolean pixel\_types=true, boolean nodata\_values=true, boolean out\_db=true , boolean extent=true);  
boolean **DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, text[] constraints);

Description

Drops PostGIS raster constraints that refer to a raster table column that were added by **AddRasterConstraints**. Useful if you need to load more data or update your raster column data. You do not need to do this if you want to get rid of a raster table or a raster column.

To drop a raster table use the standard

```
DROP TABLE mytable
```

To drop just a raster column and leave the rest of the table, use standard SQL

```
ALTER TABLE mytable DROP COLUMN rast
```

the table will disappear from the `raster_columns` catalog if the column or table is dropped. However if only the constraints are dropped, the raster column will still be listed in the `raster_columns` catalog, but there will be no other information about it aside from the column name and table.

Availability: 2.0.0

Examples

```
SELECT DropRasterConstraints ('myrasters','rast');
----RESULT output ---
t

-- verify change in raster_columns --
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types, ↵
 nodata_values
FROM raster_columns
WHERE r_table_name = 'myrasters';

srid | scale_x | scale_y | blocksize_x | blocksize_y | num_bands | pixel_types | ↵
-----+-----+-----+-----+-----+-----+-----+-----+↵
 0 | | | | | | |
```

See Also

[AddRasterConstraints](#)

10.2.3 AddOverviewConstraints

AddOverviewConstraints — Tag a raster column as being an overview of another.

## Synopsis

boolean **AddOverviewConstraints**(name ovschema, name ovtable, name ovcolumn, name refschema, name reftable, name refcolumn, int ovfactor);

boolean **AddOverviewConstraints**(name ovtable, name ovcolumn, name reftable, name refcolumn, int ovfactor);

## Description

Adds constraints on a raster column that are used to display information in the `raster_overviews` raster catalog.

The `ovfactor` parameter represents the scale multiplier in the overview column: higher overview factors have lower resolution.

When the `ovschema` and `refschema` parameters are omitted, the first table found scanning the `search_path` will be used.

Availability: 2.0.0

## Examples

```
CREATE TABLE res1 AS SELECT
ST_AddBand(
 ST_MakeEmptyRaster(1000, 1000, 0, 0, 2),
 1, '8BSI'::text, -129, NULL
) r1;

CREATE TABLE res2 AS SELECT
ST_AddBand(
 ST_MakeEmptyRaster(500, 500, 0, 0, 4),
 1, '8BSI'::text, -129, NULL
) r2;

SELECT AddOverviewConstraints('res2', 'r2', 'res1', 'r1', 2);

-- verify if registered correctly in the raster_overviews view --
SELECT o_table_name ot, o_raster_column oc,
 r_table_name rt, r_raster_column rc,
 overview_factor f
FROM raster_overviews WHERE o_table_name = 'res2';
 ot | oc | rt | rc | f
-----+-----+-----+-----+---
 res2 | r2 | res1 | r1 | 2
(1 row)
```

## See Also

Section [9.2.2](#), [DropOverviewConstraints](#), [ST\\_CreateOverview](#), [AddRasterConstraints](#)

### 10.2.4 DropOverviewConstraints

**DropOverviewConstraints** — Untag a raster column from being an overview of another.

## Synopsis

boolean **DropOverviewConstraints**(name ovschema, name ovtable, name ovcolumn);

boolean **DropOverviewConstraints**(name ovtable, name ovcolumn);

**Description**

Remove from a raster column the constraints used to show it as being an overview of another in the `raster_overviews` raster catalog.

When the `ovschema` parameter is omitted, the first table found scanning the `search_path` will be used.

Availability: 2.0.0

**See Also**

Section [9.2.2](#), [AddOverviewConstraints](#), [DropRasterConstraints](#)

**10.2.5 PostGIS\_GDAL\_Version**

`PostGIS_GDAL_Version` — Reports the version of the GDAL library in use by PostGIS.

**Synopsis**

text `PostGIS_GDAL_Version()`;

**Description**

Reports the version of the GDAL library in use by PostGIS. Will also check and report if GDAL can find its data files.

**Examples**

```
SELECT PostGIS_GDAL_Version();
 postgis_gdal_version

GDAL 1.11dev, released 2013/04/13
```

**See Also**

[postgis.gdal\\_datapath](#)

**10.2.6 PostGIS\_Raster\_Lib\_Build\_Date**

`PostGIS_Raster_Lib_Build_Date` — Reports full raster library build date.

**Synopsis**

text `PostGIS_Raster_Lib_Build_Date()`;

**Description**

Reports raster build date

## Examples

```
SELECT PostGIS_Raster_Lib_Build_Date();
postgis_raster_lib_build_date

2010-04-28 21:15:10
```

## See Also

[PostGIS\\_Raster\\_Lib\\_Version](#)

## 10.2.7 PostGIS\_Raster\_Lib\_Version

**PostGIS\_Raster\_Lib\_Version** — Reports full raster version and build configuration infos.

## Synopsis

text **PostGIS\_Raster\_Lib\_Version**();

## Description

Reports full raster version and build configuration infos.

## Examples

```
SELECT PostGIS_Raster_Lib_Version();
postgis_raster_lib_version

2.0.0
```

## See Also

[PostGIS\\_Lib\\_Version](#)

## 10.2.8 ST\_GDALDrivers

**ST\_GDALDrivers** — Returns a list of raster formats supported by PostGIS through GDAL. Only those formats with `can_write=True` can be used by `ST_AsGDALRaster`

## Synopsis

setof record **ST\_GDALDrivers**(integer OUT idx, text OUT short\_name, text OUT long\_name, text OUT can\_read, text OUT can\_write, text OUT create\_options);



## Description

Returns a list of raster formats `short_name`, `long_name` and creator options of each format supported by GDAL. Use the `short_name` as input in the `format` parameter of **ST\_AsGDALRaster**. Options vary depending on what drivers your libgdal was compiled with. `create_options` returns an xml formatted set of `CreationOptionList/Option` consisting of name and optional `type`, `description` and set of `VALUE` for each creator option for the specific driver.

Changed: 2.5.0 - add `can_read` and `can_write` columns.

Changed: 2.0.6, 2.1.3 - by default no drivers are enabled, unless GUC or Environment variable `gdal_enabled_drivers` is set.

Availability: 2.0.0 - requires GDAL >= 1.6.0.

## Examples: List of Drivers

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SELECT short_name, long_name, can_write
FROM st_gdaldrivers()
ORDER BY short_name;
```

| short_name | long_name                               | can_write |
|------------|-----------------------------------------|-----------|
| AAIGrid    | Arc/Info ASCII Grid                     | t         |
| ACE2       | ACE2                                    | f         |
| ADRG       | ARC Digitized Raster Graphics           | f         |
| AIG        | Arc/Info Binary Grid                    | f         |
| AirSAR     | AirSAR Polarimetric Image               | f         |
| ARG        | Azavea Raster Grid format               | t         |
| BAG        | Bathymetry Attributed Grid              | f         |
| BIGGIF     | Graphics Interchange Format (.gif)      | f         |
| BLX        | Magellan topo (.blx)                    | t         |
| BMP        | MS Windows Device Independent Bitmap    | f         |
| BSB        | Maptech BSB Nautical Charts             | f         |
| PAux       | PCI .aux Labelled                       | f         |
| PCIDSK     | PCIDSK Database File                    | f         |
| PCRaster   | PCRaster Raster File                    | f         |
| PDF        | Geospatial PDF                          | f         |
| PDS        | NASA Planetary Data System              | f         |
| PDS4       | NASA Planetary Data System 4            | t         |
| PLMOSAIC   | Planet Labs Mosaics API                 | f         |
| PLSCENES   | Planet Labs Scenes API                  | f         |
| PNG        | Portable Network Graphics               | t         |
| PNM        | Portable Pixmap Format (netpbm)         | f         |
| PRF        | Racurs PHOTOMOD PRF                     | f         |
| R          | R Object Data Store                     | t         |
| Rasterlite | Rasterlite                              | t         |
| RDA        | DigitalGlobe Raster Data Access driver  | f         |
| RIK        | Swedish Grid RIK (.rik)                 | f         |
| RMF        | Raster Matrix Format                    | f         |
| ROI_PAC    | ROI_PAC raster                          | f         |
| RPFTOC     | Raster Product Format TOC format        | f         |
| RRASTER    | R Raster                                | f         |
| RS2        | RadarSat 2 XML Product                  | f         |
| RST        | Idrisi Raster A.1                       | t         |
| SAFE       | Sentinel-1 SAR SAFE Product             | f         |
| SAGA       | SAGA GIS Binary Grid (.sdat, .sg-grd-z) | t         |
| SAR_CEOS   | CEOS SAR Image                          | f         |
| SDTS       | SDTS Raster                             | f         |
| SENTINEL2  | Sentinel 2                              | f         |
| SGI        | SGI Image File Format 1.0               | f         |
| SNODAS     | Snow Data Assimilation System           | f         |
| SRP        | Standard Raster Product (ASRP/USRP)     | f         |

|          |                                    |   |
|----------|------------------------------------|---|
| SRTMHGT  | SRTMHGT File Format                | t |
| Terragen | Terragen heightfield               | f |
| TIL      | EarthWatch .TIL                    | f |
| TSX      | TerraSAR-X Product                 | f |
| USGSDEM  | USGS Optional ASCII DEM (and CDED) | t |
| VICAR    | MIPL VICAR file                    | f |
| VRT      | Virtual Raster                     | t |
| WCS      | OGC Web Coverage Service           | f |
| WMS      | OGC Web Map Service                | t |
| WMTS     | OGC Web Map Tile Service           | t |
| XPM      | X11 PixMap Format                  | t |
| XYZ      | ASCII Gridded XYZ                  | t |
| ZMap     | ZMap Plus Grid                     | t |

### Example: List of options for each driver

```
-- Output the create options XML column of JPEG as a table --
-- Note you can use these creator options in ST_AsGDALRaster options argument
SELECT (xpath('@name', g.opt))[1]::text As oname,
 (xpath('@type', g.opt))[1]::text As otype,
 (xpath('@description', g.opt))[1]::text As descrip
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM st_gdaldrivers()
WHERE short_name = 'JPEG') As g;
```

| oname              | otype   | descrip                                                                                      |
|--------------------|---------|----------------------------------------------------------------------------------------------|
| PROGRESSIVE        | boolean | whether to generate a progressive JPEG                                                       |
| QUALITY            | int     | good=100, bad=0, default=75                                                                  |
| WORLDFILE          | boolean | whether to generate a worldfile                                                              |
| INTERNAL_MASK      | boolean | whether to generate a validity mask                                                          |
| COMMENT            | string  | Comment                                                                                      |
| SOURCE_ICC_PROFILE | string  | ICC profile encoded in Base64                                                                |
| EXIF_THUMBNAIL     | boolean | whether to generate an EXIF thumbnail(overview).<br>By default its max dimension will be 128 |
| THUMBNAIL_WIDTH    | int     | Forced thumbnail width                                                                       |
| THUMBNAIL_HEIGHT   | int     | Forced thumbnail height                                                                      |
| (9 rows)           |         |                                                                                              |

```
-- raw xml output for creator options for GeoTiff --
SELECT create_options
FROM st_gdaldrivers()
WHERE short_name = 'GTiff';
```

```
<CreationOptionList>
 <Option name="COMPRESS" type="string-select">
 <Value>
>NONE</Value>
 <Value>
>LZW</Value>
 <Value>
>PACKBITS</Value>
 <Value>
>JPEG</Value>
 <Value>
>CCITTRLE</Value>
 <Value>
>CCITTFAX3</Value>
 <Value>
>CCITTFAX4</Value>
```

```

 <Value>
>DEFLATE</Value>
 </Option>
 <Option name="PREDICTOR" type="int" description="Predictor Type"/>
 <Option name="JPEG_QUALITY" type="int" description="JPEG quality 1-100" default="75"/>
 <Option name="ZLEVEL" type="int" description="DEFLATE compression level 1-9" default ←
 ="6"/>
 <Option name="NBITS" type="int" description="BITS for sub-byte files (1-7), sub-uint16 ←
 (9-15), sub-uint32 (17-31)"/>
 <Option name="INTERLEAVE" type="string-select" default="PIXEL">
 <Value>
>BAND</Value>
 <Value>
>PIXEL</Value>
 </Option>
 <Option name="TILED" type="boolean" description="Switch to tiled format"/>
 <Option name="TFW" type="boolean" description="Write out world file"/>
 <Option name="RPB" type="boolean" description="Write out .RPB (RPC) file"/>
 <Option name="BLOCKXSIZE" type="int" description="Tile Width"/>
 <Option name="BLOCKYSIZE" type="int" description="Tile/Strip Height"/>
 <Option name="PHOTOMETRIC" type="string-select">
 <Value>
>MINISBLACK</Value>
 <Value>
>MINISWHITE</Value>
 <Value>
>PALETTE</Value>
 <Value>
>RGB</Value>
 <Value>
>CMYK</Value>
 <Value>
>YCBCR</Value>
 <Value>
>CIELAB</Value>
 <Value>
>ICCLAB</Value>
 <Value>
>ITULAB</Value>
 </Option>
 <Option name="SPARSE_OK" type="boolean" description="Can newly created files have ←
 missing blocks?" default="FALSE"/>
 <Option name="ALPHA" type="boolean" description="Mark first extrasample as being alpha ←
 "/>
 <Option name="PROFILE" type="string-select" default="GDALGeoTIFF">
 <Value>
>GDALGeoTIFF</Value>
 <Value>
>GeoTIFF</Value>
 <Value>
>BASELINE</Value>
 </Option>
 <Option name="PIXELTYPE" type="string-select">
 <Value>
>DEFAULT</Value>
 <Value>
>SIGNEDBYTE</Value>
 </Option>
 <Option name="BIGTIFF" type="string-select" description="Force creation of BigTIFF file ←
 ">
 <Value>
>YES</Value>

```

```

 <Value
>NO</Value>
 <Value
>IF_NEEDED</Value>
 <Value
>IF_SAFER</Value>
 </Option>
 <Option name="ENDIANNESS" type="string-select" default="NATIVE" description="Force ↵
 endianness of created file. For DEBUG purpose mostly">
 <Value
>NATIVE</Value>
 <Value
>INVERTED</Value>
 <Value
>LITTLE</Value>
 <Value
>BIG</Value>
 </Option>
 <Option name="COPY_SRC_OVERVIEWS" type="boolean" default="NO" description="Force copy ↵
 of overviews of source dataset (CreateCopy())"/>
</CreationOptionList
>

```

```
-- Output the create options XML column for GTiff as a table --
```

```

SELECT (xpath('@name', g.opt))[1]::text As oname,
 (xpath('@type', g.opt))[1]::text As otype,
 (xpath('@description', g.opt))[1]::text As descrip,
 array_to_string(xpath('Value/text()', g.opt), ', ') As vals
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM st_gdaldrivers()
WHERE short_name = 'GTiff') As g;

```

| oname                                                         | otype         | descrip                                                              | vals          |
|---------------------------------------------------------------|---------------|----------------------------------------------------------------------|---------------|
| COMPRESS                                                      | string-select |                                                                      | NONE, LZW, ↵  |
| PACKBITS, JPEG, CCITTRLE, CCITTFAX3, CCITTFAX4, DEFLATE       |               |                                                                      |               |
| PREDICTOR                                                     | int           | Predictor Type                                                       | ↵             |
| JPEG_QUALITY                                                  | int           | JPEG quality 1-100                                                   | ↵             |
| ZLEVEL                                                        | int           | DEFLATE compression level 1-9                                        | ↵             |
| NBITS                                                         | int           | BITS for sub-byte files (1-7), sub-uint16 (9-15), sub-uint32 (17-31) | ↵             |
| INTERLEAVE                                                    | string-select |                                                                      | BAND, PIXEL   |
| TILED                                                         | boolean       | Switch to tiled format                                               | ↵             |
| TFW                                                           | boolean       | Write out world file                                                 | ↵             |
| RPB                                                           | boolean       | Write out .RPB (RPC) file                                            | ↵             |
| BLOCKXSIZE                                                    | int           | Tile Width                                                           | ↵             |
| BLOCKYSIZE                                                    | int           | Tile/Strip Height                                                    | ↵             |
| PHOTOMETRIC                                                   | string-select |                                                                      | MINISBLACK, ↵ |
| MINISWHITE, PALETTE, RGB, CMYK, YCBCR, CIELAB, ICCLAB, ITULAB |               |                                                                      |               |

|                    |                               |                                                         |
|--------------------|-------------------------------|---------------------------------------------------------|
| SPARSE_OK          | boolean                       | Can newly created files have missing blocks? ↵          |
|                    |                               |                                                         |
| ALPHA              | boolean                       | Mark first extrasample as being alpha ↵                 |
|                    |                               |                                                         |
| PROFILE            | string-select                 | ↵                                                       |
|                    |                               | GDALGeoTIFF, ↵                                          |
| GeoTIFF, BASELINE  |                               |                                                         |
| PIXELTYPE          | string-select                 | ↵                                                       |
|                    |                               | DEFAULT, ↵                                              |
| SIGNEDBYTE         |                               |                                                         |
| BIGTIFF            | string-select                 | Force creation of BigTIFF file ↵                        |
|                    |                               | YES, NO, IF_NEEDED, IF_SAFER                            |
| ENDIANNESS         | string-select                 | Force endianness of created file. For DEBUG purpose ↵   |
| mostly             | NATIVE, INVERTED, LITTLE, BIG |                                                         |
| COPY_SRC_OVERVIEWS | boolean                       | Force copy of overviews of source dataset (CreateCopy ↵ |
| ( ))               |                               |                                                         |
| (19 rows)          |                               |                                                         |

See Also

[ST\\_AsGDALRaster](#), [ST\\_SRID](#), [postgis.gdal\\_enabled\\_drivers](#)

10.2.9 ST\_Contour

ST\_Contour — Generates a set of vector contours from the provided raster band, using the [GDAL contouring algorithm](#).

Synopsis

setof record **ST\_Contour**(raster rast, integer bandnumber, double precision level\_interval, double precision level\_base, double precision[] fixed\_levels, boolean polygonize);

Description

Generates a set of vector contours from the provided raster band, using the [GDAL contouring algorithm](#).  
When the `fixed_levels` parameter is a non-empty array, the `level_interval` and `level_base` parameters are ignored.  
Return values are a set of records with the following attributes:

- geom** The geometry of the contour line.
- id** A unique identifier given to the contour line by GDAL.
- value** The raster value the line represents. For an elevation DEM input, this would be the elevation of the output contour.

Availability: 3.2.0

Example

```
WITH c AS (
SELECT (ST_Contour(rast, 1, fixed_levels =
> ARRAY[100.0, 200.0, 300.0])).*
FROM dem_grid WHERE rid = 1
)
SELECT st_astext(geom), id, value
FROM c;
```

**See Also**[ST\\_InterpolateRaster](#)**10.2.10 ST\_InterpolateRaster**

**ST\_InterpolateRaster** — Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation.

**Synopsis**

```
raster ST_InterpolateRaster(geometry input_points, text algorithm_options, raster template, integer template_band_num=1);
```

**Description**

Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation. There are five interpolation algorithms available: inverse distance, inverse distance nearest-neighbor, moving average, nearest neighbor, and linear interpolation. See the [gdal\\_grid documentation](#) for more details on the algorithms and their parameters. For more information on how interpolations are calculated, see the [GDAL grid tutorial](#).

Input parameters are:

**input\_points** The points to drive the interpolation. Any geometry with Z-values is acceptable, all points in the input will be used.

**algorithm\_options** A string defining the algorithm and algorithm options, in the format used by [gdal\\_grid](#). For example, for an inverse-distance interpolation with a smoothing of 2, you would use "invdist:smoothing=2.0"

**template** A raster template to drive the geometry of the output raster. The width, height, pixel size, spatial extent and pixel type will be read from this template.

**template\_band\_num** By default the first band in the template raster is used to drive the output raster, but that can be adjusted with this parameter.

Availability: 3.2.0

**Example**

```
SELECT ST_InterpolateRaster(
 'MULTIPOINT(10.5 9.5 1000, 11.5 8.5 1000, 10.5 8.5 500, 11.5 9.5 500) '::geometry,
 'invdist:smoothing:2.0',
 ST_AddBand(ST_MakeEmptyRaster(200, 400, 10, 10, 0.01, -0.005, 0, 0), '16BSI')
)
```

**See Also**[ST\\_Contour](#)**10.2.11 UpdateRasterSRID**

**UpdateRasterSRID** — Change the SRID of all rasters in the user-specified column and table.

## Synopsis

raster **UpdateRasterSRID**(name schema\_name, name table\_name, name column\_name, integer new\_srid);  
 raster **UpdateRasterSRID**(name table\_name, name column\_name, integer new\_srid);

## Description

Change the SRID of all rasters in the user-specified column and table. The function will drop all appropriate column constraints (extent, alignment and SRID) before changing the SRID of the specified column's rasters.



### Note

The data (band pixel values) of the rasters are not touched by this function. Only the raster's metadata is changed.

Availability: 2.1.0

## See Also

[UpdateGeometrySRID](#)

## 10.2.12 ST\_CreateOverview

**ST\_CreateOverview** — Create an reduced resolution version of a given raster coverage.

## Synopsis

regclass **ST\_CreateOverview**(regclass tab, name col, int factor, text algo='NearestNeighbor');

## Description

Create an overview table with resampled tiles from the source table. Output tiles will have the same size of input tiles and cover the same spatial extent with a lower resolution (pixel size will be  $1/\text{factor}$  of the original in both directions).

The overview table will be made available in the `raster_overviews` catalog and will have raster constraints enforced.

Algorithm options are: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', and 'Lanczos'. Refer to: [GDAL Warp resampling methods](#) for more details.

Availability: 2.2.0

## Example

Output to generally better quality but slower to product format

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2, 'Lanczos');
```

Output to faster to process default nearest neighbor

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2);
```

## See Also

[ST\\_Retile](#), [AddOverviewConstraints](#), [AddRasterConstraints](#), [Section 9.2.2](#)

## 10.3 Raster Constructors

### 10.3.1 ST\_AddBand

**ST\_AddBand** — Returns a raster with the new band(s) of given type added with given initial value in the given index location. If no index is specified, the band is added to the end.

#### Synopsis

- (1) raster **ST\_AddBand**(raster rast, addbandarg[] addbandargset);
- (2) raster **ST\_AddBand**(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
- (3) raster **ST\_AddBand**(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
- (4) raster **ST\_AddBand**(raster torast, raster fromrast, integer fromband=1, integer torastindex=at\_end);
- (5) raster **ST\_AddBand**(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at\_end);
- (6) raster **ST\_AddBand**(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);
- (7) raster **ST\_AddBand**(raster rast, text outdbfile, integer[] outdbindex, integer index=at\_end, double precision nodataval=NULL);

#### Description

Returns a raster with a new band added in given position (index), of given type, of given initial value, and of given nodata value. If no index is specified, the band is added to the end. If no `fromband` is specified, band 1 is assumed. Pixel type is a string representation of one of the pixel types specified in [ST\\_BandPixelType](#). If an existing index is specified all subsequent bands  $\geq$  that index are incremented by 1. If an initial value greater than the max of the pixel type is specified, then the initial value is set to the highest value allowed by the pixel type.

For the variant that takes an array of `addbandarg` (Variant 1), a specific `addbandarg`'s index value is relative to the raster at the time when the band described by that `addbandarg` is being added to the raster. See the Multiple New Bands example below.

For the variant that takes an array of rasters (Variant 5), if `torast` is NULL then the `fromband` band of each raster in the array is accumulated into a new raster.

For the variants that take `outdbfile` (Variants 6 and 7), the value must include the full path to the raster file. The file must also be accessible to the postgres server process.

Enhanced: 2.1.0 support for `addbandarg` added.

Enhanced: 2.1.0 support for new out-db bands added.

#### Examples: Single New Band

```
-- Add another band of type 8 bit unsigned integer with pixels initialized to 200
UPDATE dummy_rast
SET rast = ST_AddBand(rast, '8BUI'::text, 200)
WHERE rid = 1;
```

```
-- Create an empty raster 100x100 units, with upper left right at 0, add 2 bands (band 1 ←
 is 0/1 boolean bit switch, band2 allows values 0-15)
-- uses addbandargs
INSERT INTO dummy_rast(rid, rast)
VALUES (10, ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 1, -1, 0, 0, 0),
 ARRAY[
 ROW(1, '1BB'::text, 0, NULL),
 ROW(2, '4BUI'::text, 0, NULL)
]::addbandarg[]
)
);
```

```
-- output meta data of raster bands to verify all is right --
```



```
SELECT (bmd).*
FROM (SELECT ST_BandMetaData(rast,generate_series(1,2)) As bmd
 FROM dummy_rast WHERE rid = 10) AS foo;
--result --
pixeltype | nodatavalue | isoutdb | path
-----+-----+-----+-----
1BB | | f |
4BUI | | f |

-- output meta data of raster -
SELECT (rmd).width, (rmd).height, (rmd).numbands
FROM (SELECT ST_MetaData(rast) As rmd
 FROM dummy_rast WHERE rid = 10) AS foo;
-- result --
upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
0 | 0 | 100 | 100 | 1 | -1 | 0 | 0 | 0 | 2 |
```

Examples: Multiple New Bands

```
SELECT
 *
FROM ST_BandMetadata(
 ST_AddBand(
 ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
 ARRAY[
 ROW(NULL, '8BUI', 255, 0),
 ROW(NULL, '16BUI', 1, 2),
 ROW(2, '32BUI', 100, 12),
 ROW(2, '32BF', 3.14, -1)
]::addbandarg[]
),
 ARRAY[]::integer[]
);

bandnum | pixeltype | nodatavalue | isoutdb | path
-----+-----+-----+-----+-----
1 | 8BUI | 0 | f |
2 | 32BF | -1 | f |
3 | 32BUI | 12 | f |
4 | 16BUI | 2 | f |
```

```
-- Aggregate the 1st band of a table of like rasters into a single raster
-- with as many bands as there are test_types and as many rows (new rasters) as there are mice
-- NOTE: The ORDER BY test_type is only supported in PostgreSQL 9.0+
-- for 8.4 and below it usually works to order your data in a subselect (but not guaranteed)
-- The resulting raster will have a band for each test_type alphabetical by test_type
-- For mouse lovers: No mice were harmed in this exercise
SELECT
 mouse,
 ST_AddBand(NULL, array_agg(rast ORDER BY test_type), 1) As rast
FROM mice_studies
GROUP BY mouse;
```

**Examples: New Out-db band**

```

SELECT
 *
FROM ST_BandMetadata (
 ST_AddBand(
 ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
 '/home/raster/mytestraster.tif'::text, NULL::int[]
),
 ARRAY[]::integer[]
);

```

| bandnum | pixeltype | nodatavalue | isoutdb | path                          |
|---------|-----------|-------------|---------|-------------------------------|
| 1       | 8BUI      |             | t       | /home/raster/mytestraster.tif |
| 2       | 8BUI      |             | t       | /home/raster/mytestraster.tif |
| 3       | 8BUI      |             | t       | /home/raster/mytestraster.tif |

**See Also**

[ST\\_BandMetaData](#), [ST\\_BandPixelType](#), [ST\\_MakeEmptyRaster](#), [ST\\_MetaData](#), [ST\\_NumBands](#), [ST\\_Reclass](#)

**10.3.2 ST\_AsRaster**

**ST\_AsRaster** — Converts a PostGIS geometry to a PostGIS raster.

**Synopsis**

raster **ST\_AsRaster**(geometry geom, raster ref, text pixeltype, double precision value=1, double precision nodataval=0, boolean touched=false);

raster **ST\_AsRaster**(geometry geom, raster ref, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], boolean touched=false);

raster **ST\_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST\_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST\_AsRaster**(geometry geom, double precision scalex, double precision scaley, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST\_AsRaster**(geometry geom, double precision scalex, double precision scaley, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST\_AsRaster**(geometry geom, integer width, integer height, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST\_AsRaster**(geometry geom, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST\_AsRaster**(geometry geom, integer width, integer height, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST\_AsRaster**(geometry geom, integer width, integer height, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

## Description

Converts a PostGIS geometry to a PostGIS raster. The many variants offers three groups of possibilities for setting the alignment and pixel size of the resulting raster.

The first group, composed of the two first variants, produce a raster having the same alignment (`scalex`, `scaley`, `gridx` and `gridy`), pixel type and nodata value as the provided reference raster. You generally pass this reference raster by joining the table containing the geometry with the table containing the reference raster.

The second group, composed of four variants, let you set the dimensions of the raster by providing the parameters of a pixel size (`scalex` & `scaley` and `skewx` & `skewy`). The width & height of the resulting raster will be adjusted to fit the extent of the geometry. In most cases, you must cast integer `scalex` & `scaley` arguments to double precision so that PostgreSQL choose the right variant.

The third group, composed of four variants, let you fix the dimensions of the raster by providing the dimensions of the raster (width & height). The parameters of the pixel size (`scalex` & `scaley` and `skewx` & `skewy`) of the resulting raster will be adjusted to fit the extent of the geometry.

The two first variants of each of those two last groups let you specify the alignment with an arbitrary corner of the alignment grid (`gridx` & `gridy`) and the two last variants takes the upper left corner (`upperleftx` & `upperlefty`).

Each group of variant allows producing a one band raster or a multiple bands raster. To produce a multiple bands raster, you must provide an array of pixel types (`pixeltype[]`), an array of initial values (`value`) and an array of nodata values (`nodataval`). If not provided `pixeltype` defaults to 8BUI, values to 1 and `nodataval` to 0.

The output raster will be in the same spatial reference as the source geometry. The only exception is for variants with a reference raster. In this case the resulting raster will get the same SRID as the reference raster.

The optional `touched` parameter defaults to false and maps to the GDAL `ALL_TOUCHED` rasterization option, which determines if pixels touched by lines or polygons will be burned. Not just those on the line render path, or whose center point is within the polygon.

This is particularly useful for rendering jpegs and pngs of geometries directly from the database when using in combination with [ST\\_AsPNG](#) and other [ST\\_AsGDALRaster](#) family of functions.

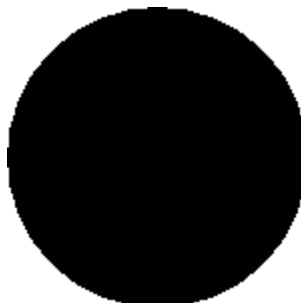
Availability: 2.0.0 - requires GDAL >= 1.6.0.



### Note

Not yet capable of rendering complex geometry types such as curves, TINS, and PolyhedralSurfaces, but should be able too once GDAL can.

## Examples: Output geometries as PNG files



*black circle*

```
-- this will output a black circle taking up 150 x 150 pixels --
SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10),150, 150));
```



*example from buffer rendered with just PostGIS*

```
-- the bands map to RGB bands - the value (118,154,118) - teal --
SELECT ST_AsPNG(
 ST_AsRaster(
 ST_Buffer(
 ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10,'join=bevel'),
 200,200,ARRAY['8BUI', '8BUI', '8BUI'], ARRAY[118,154,118], ARRAY[0,0,0]));
```

#### See Also

[ST\\_BandPixelType](#), [ST\\_Buffer](#), [ST\\_GDALDrivers](#), [ST\\_AsGDALRaster](#), [ST\\_AsPNG](#), [ST\\_AsJPEG](#), [ST\\_SRID](#)

### 10.3.3 ST\_Band

**ST\_Band** — Returns one or more bands of an existing raster as a new raster. Useful for building new rasters from existing rasters.

#### Synopsis

```
raster ST_Band(raster rast, integer[] nbands = ARRAY[1]);
raster ST_Band(raster rast, integer nband);
raster ST_Band(raster rast, text nbands, character delimiter=,);
```

#### Description

Returns one or more bands of an existing raster as a new raster. Useful for building new rasters from existing rasters or export of only selected bands of a raster or rearranging the order of bands in a raster. If no band is specified or any of specified bands does not exist in the raster, then all bands are returned. Used as a helper function in various functions such as for deleting a band.



#### Warning

For the `nbands` as text variant of function, the default delimiter is `,` which means you can ask for `'1,2,3'` and if you wanted to use a different delimiter you would do `ST_Band(rast, '1@2@3', '@')`. For asking for multiple bands, we strongly suggest you use the array form of this function e.g. `ST_Band(rast, '{1,2,3}'::int[])`; since the `text` list of bands form may be removed in future versions of PostGIS.

Availability: 2.0.0

Examples

```
-- Make 2 new rasters: 1 containing band 1 of dummy, second containing band 2 of dummy and then reclassified as a 2BUI
SELECT ST_NumBands(rast1) As numb1, ST_BandPixelType(rast1) As pix1,
 ST_NumBands(rast2) As numb2, ST_BandPixelType(rast2) As pix2
FROM (
 SELECT ST_Band(rast) As rast1, ST_Reclass(ST_Band(rast,3), '100-200):1, [200-254:2', '2 BUI') As rast2
 FROM dummy_rast
 WHERE rid = 2) As foo;
```

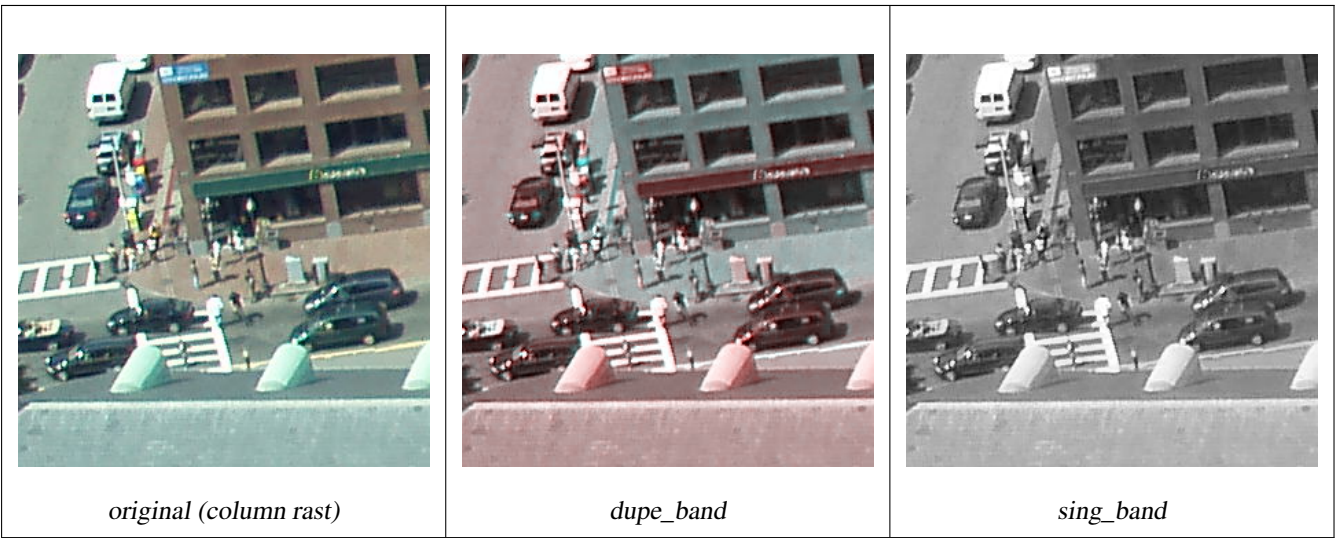
| numb1 | pix1 | numb2 | pix2 |
|-------|------|-------|------|
| 1     | 8BUI | 1     | 2BUI |

```
-- Return bands 2 and 3. Using array cast syntax
SELECT ST_NumBands(ST_Band(rast, '{2,3}'::int[])) As num_bands
FROM dummy_rast WHERE rid=2;
```

```
num_bands

2
```

```
-- Return bands 2 and 3. Use array to define bands
SELECT ST_NumBands(ST_Band(rast, ARRAY[2,3])) As num_bands
FROM dummy_rast
WHERE rid=2;
```



```
--Make a new raster with 2nd band of original and 1st band repeated twice,
and another with just the third band
SELECT rast, ST_Band(rast, ARRAY[2,1,1]) As dupe_band,
 ST_Band(rast, 3) As sing_band
FROM samples.than_chunked
WHERE rid=35;
```

See Also

ST\_AddBand, ST\_NumBands, ST\_Reclass, Chapter 10

### 10.3.4 ST\_MakeEmptyCoverage

ST\_MakeEmptyCoverage — Cover georeferenced area with a grid of empty raster tiles.

#### Synopsis

raster **ST\_MakeEmptyCoverage**(integer tilewidth, integer tileheight, integer width, integer height, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy, integer srid=unknown);

#### Description

Create a set of raster tiles with **ST\_MakeEmptyRaster**. Grid dimension is width & height. Tile dimension is tilewidth & tileheight. The covered georeferenced area is from upper left corner (upperleftx, upperlefty) to lower right corner (upperleftx + width \* scalex, upperlefty + height \* scaley).



**Note**  
Note that scaley is generally negative for rasters and scalex is generally positive. So lower right corner will have a lower y value and higher x value than the upper left corner.

Availability: 2.4.0

#### Examples Basic

Create 16 tiles in a 4x4 grid to cover the WGS84 area from upper left corner (22, 77) to lower right corner (55, 33).

SELECT (ST\_MetaData(tile)).\* FROM ST\_MakeEmptyCoverage(1, 1, 4, 4, 22, 33, (55 - 22)/(4)::float, (33 - 77)/(4)::float, 0., 0., 4326) tile; ↵

| upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | ↵ |
|------------|------------|-------|--------|--------|--------|-------|-------|------|---|
| numbands   |            |       |        |        |        |       |       |      |   |
| 22         | 33         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | ↵ |
|            | 0          |       |        |        |        |       |       |      |   |
| 30.25      | 33         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | ↵ |
|            | 0          |       |        |        |        |       |       |      |   |
| 38.5       | 33         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | ↵ |
|            | 0          |       |        |        |        |       |       |      |   |
| 46.75      | 33         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | ↵ |
|            | 0          |       |        |        |        |       |       |      |   |
| 22         | 22         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | ↵ |
|            | 0          |       |        |        |        |       |       |      |   |
| 30.25      | 22         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | ↵ |
|            | 0          |       |        |        |        |       |       |      |   |
| 38.5       | 22         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | ↵ |
|            | 0          |       |        |        |        |       |       |      |   |
| 46.75      | 22         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | ↵ |
|            | 0          |       |        |        |        |       |       |      |   |
| 22         | 11         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | ↵ |
|            | 0          |       |        |        |        |       |       |      |   |
| 30.25      | 11         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | ↵ |
|            | 0          |       |        |        |        |       |       |      |   |
| 38.5       | 11         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | ↵ |
|            | 0          |       |        |        |        |       |       |      |   |
| 46.75      | 11         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | ↵ |
|            | 0          |       |        |        |        |       |       |      |   |

|       |  |   |  |   |  |   |  |      |  |     |  |   |  |   |  |      |  |   |
|-------|--|---|--|---|--|---|--|------|--|-----|--|---|--|---|--|------|--|---|
| 22    |  | 0 |  | 1 |  | 1 |  | 8.25 |  | -11 |  | 0 |  | 0 |  | 4326 |  | ↔ |
|       |  | 0 |  |   |  |   |  |      |  |     |  |   |  |   |  |      |  |   |
| 30.25 |  | 0 |  | 1 |  | 1 |  | 8.25 |  | -11 |  | 0 |  | 0 |  | 4326 |  | ↔ |
|       |  | 0 |  |   |  |   |  |      |  |     |  |   |  |   |  |      |  |   |
| 38.5  |  | 0 |  | 1 |  | 1 |  | 8.25 |  | -11 |  | 0 |  | 0 |  | 4326 |  | ↔ |
|       |  | 0 |  |   |  |   |  |      |  |     |  |   |  |   |  |      |  |   |
| 46.75 |  | 0 |  | 1 |  | 1 |  | 8.25 |  | -11 |  | 0 |  | 0 |  | 4326 |  | ↔ |
|       |  | 0 |  |   |  |   |  |      |  |     |  |   |  |   |  |      |  |   |

See Also

[ST\\_MakeEmptyRaster](#)

10.3.5 ST\_MakeEmptyRaster

ST\_MakeEmptyRaster — Returns an empty raster (having no bands) of given dimensions (width & height), upperleft X and Y, pixel size and rotation (scalex, scaley, skewx & skewy) and reference system (srid). If a raster is passed in, returns a new raster with the same size, alignment and SRID. If srid is left out, the spatial ref is set to unknown (0).

Synopsis

raster **ST\_MakeEmptyRaster**(raster rast);  
raster **ST\_MakeEmptyRaster**(integer width, integer height, float8 upperleftx, float8 upperlefty, float8 scalex, float8 scaley, float8 skewx, float8 skewy, integer srid=unknown);  
raster **ST\_MakeEmptyRaster**(integer width, integer height, float8 upperleftx, float8 upperlefty, float8 pixelsize);

Description

Returns an empty raster (having no band) of given dimensions (width & height) and georeferenced in spatial (or world) coordinates with upper left X (upperleftx), upper left Y (upperlefty), pixel size and rotation (scalex, scaley, skewx & skewy) and reference system (srid).

The last version use a single parameter to specify the pixel size (pixelsize). scalex is set to this argument and scaley is set to the negative value of this argument. skewx and skewy are set to 0.

If an existing raster is passed in, it returns a new raster with the same meta data settings (without the bands).

If no srid is specified it defaults to 0. After you create an empty raster you probably want to add bands to it and maybe edit it. Refer to [ST\\_AddBand](#) to define bands and [ST\\_SetValue](#) to set initial pixel values.

Examples

```
INSERT INTO dummy_rast(rid,rast)
VALUES(3, ST_MakeEmptyRaster(100, 100, 0.0005, 0.0005, 1, 1, 0, 0, 4326));

--use an existing raster as template for new raster
INSERT INTO dummy_rast(rid,rast)
SELECT 4, ST_MakeEmptyRaster(rast)
FROM dummy_rast WHERE rid = 3;

-- output meta data of rasters we just added
SELECT rid, (md).*
FROM (SELECT rid, ST_MetaData(rast) As md
 FROM dummy_rast
 WHERE rid IN(3,4)) As foo;
```

```
-- output --
rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | ↵
numbands
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
3 | 0.0005 | 0.0005 | 100 | 100 | 1 | 1 | 0 | 0 | ↵
4326 | 0
4 | 0.0005 | 0.0005 | 100 | 100 | 1 | 1 | 0 | 0 | ↵
4326 | 0
```

## See Also

ST\_AddBand, ST\_MetaData, ST\_ScaleX, ST\_ScaleY, ST\_SetValue, ST\_SkewX, , ST\_SkewY

### 10.3.6 ST\_Tile

**ST\_Tile** — Returns a set of rasters resulting from the split of the input raster based upon the desired dimensions of the output rasters.

## Synopsis

```
setof raster ST_Tile(raster rast, int[] nband, integer width, integer height, boolean padwithnodata=FALSE, double precision no-
dataval=NULL);
```

```
setof raster ST_Tile(raster rast, integer nband, integer width, integer height, boolean padwithnodata=FALSE, double precision
nodataval=NULL);
```

```
setof raster ST_Tile(raster rast, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);
```

### Description

Returns a set of rasters resulting from the split of the input raster based upon the desired dimensions of the output rasters.

If `padwithnodata = FALSE`, edge tiles on the right and bottom sides of the raster may have different dimensions than the rest of the tiles. If `padwithnodata = TRUE`, all tiles will have the same dimensions with the possibility that edge tiles being padded with `NODATA` values. If raster band(s) do not have `NODATA` value(s) specified, one can be specified by setting `nodataval`.



### Note

If a specified band of the input raster is out-of-db, the corresponding band in the output rasters will also be out-of-db.

Availability: 2.1.0

## Examples

```
WITH foo AS (
 SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', 1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
 SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', 2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
 SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', 3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL
```



```

SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
 SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
 SELECT ST_Tile(rast, 3, 3, TRUE) AS rast FROM bar
)
SELECT
 ST_DumpValues(rast)
FROM baz;

```

st\_dumpvalues

```

(1,"{{1,1,1},{1,1,1},{1,1,1}}")
(2,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{2,2,2},{2,2,2},{2,2,2}}")
(2,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{3,3,3},{3,3,3},{3,3,3}}")
(2,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{4,4,4},{4,4,4},{4,4,4}}")
(2,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{5,5,5},{5,5,5},{5,5,5}}")
(2,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{6,6,6},{6,6,6},{6,6,6}}")
(2,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{7,7,7},{7,7,7},{7,7,7}}")
(2,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{8,8,8},{8,8,8},{8,8,8}}")
(2,"{{80,80,80},{80,80,80},{80,80,80}}")
(1,"{{9,9,9},{9,9,9},{9,9,9}}")
(2,"{{90,90,90},{90,90,90},{90,90,90}}")
(18 rows)

```

```

WITH foo AS (
 SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
 SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
 SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

 SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
 SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
 SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

 SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
 SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
 SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 9, 0), 2, '8BUI', 90, 0) AS rast UNION ALL
)
SELECT ST_DumpValues(rast)
FROM foo;

```

```

 ', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
 SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI', 80, 0), 2, '8BUI', 80, 0) AS rast
), bar AS (
 SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
 SELECT ST_Tile(rast, 3, 3, 2) AS rast FROM bar
)
SELECT
 ST_DumpValues(rast)
FROM baz;

```

| st_dumpvalues                            |
|------------------------------------------|
| (1, "{10,10,10},{10,10,10},{10,10,10}}") |
| (1, "{20,20,20},{20,20,20},{20,20,20}}") |
| (1, "{30,30,30},{30,30,30},{30,30,30}}") |
| (1, "{40,40,40},{40,40,40},{40,40,40}}") |
| (1, "{50,50,50},{50,50,50},{50,50,50}}") |
| (1, "{60,60,60},{60,60,60},{60,60,60}}") |
| (1, "{70,70,70},{70,70,70},{70,70,70}}") |
| (1, "{80,80,80},{80,80,80},{80,80,80}}") |
| (1, "{90,90,90},{90,90,90},{90,90,90}}") |

(9 rows)

**See Also**

[ST\\_Union](#), [ST\\_Retile](#)

**10.3.7 ST\_Retile**

**ST\_Retile** — Return a set of configured tiles from an arbitrarily tiled raster coverage.

**Synopsis**

SETOF raster **ST\_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

**Description**

Return a set of tiles having the specified scale (*sfx*, *sfy*) and max size (*tw*, *th*) and covering the specified extent (*ext*) with data coming from the specified raster coverage (*tab*, *col*).

Algorithm options are: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', and 'Lanczos'. Refer to: [GDAL Warp resampling methods](#) for more details.

Availability: 2.2.0

**See Also**

[ST\\_CreateOverview](#)

**10.3.8 ST\_FromGDALRaster**

**ST\_FromGDALRaster** — Returns a raster from a supported GDAL raster file.

Synopsis

raster **ST\_FromGDALRaster**(bytea gdaldata, integer srid=NULL);

Description

Returns a raster from a supported GDAL raster file. `gdaldata` is of type `bytea` and should be the contents of the GDAL raster file.

If `srid` is `NULL`, the function will try to automatically assign the SRID from the GDAL raster. If `srid` is provided, the value provided will override any automatically assigned SRID.

Availability: 2.1.0

Examples

```
WITH foo AS (
 SELECT ST_AsPNG(ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.1, ←
 -0.1, 0, 0, 4326), 1, '8BUI', 1, 0), 2, '8BUI', 2, 0), 3, '8BUI', 3, 0)) AS png
),
bar AS (
 SELECT 1 AS rid, ST_FromGDALRaster(png) AS rast FROM foo
 UNION ALL
 SELECT 2 AS rid, ST_FromGDALRaster(png, 3310) AS rast FROM foo
)
SELECT
 rid,
 ST_Metadatas(rast) AS metadata,
 ST_SummaryStats(rast, 1) AS stats1,
 ST_SummaryStats(rast, 2) AS stats2,
 ST_SummaryStats(rast, 3) AS stats3
FROM bar
ORDER BY rid;
```

| rid | metadata                  | stats1        | stats2        | stats3         |
|-----|---------------------------|---------------|---------------|----------------|
| 1   | (0,0,2,2,1,-1,0,0,0,3)    | (4,4,1,0,1,1) | (4,8,2,0,2,2) | (4,12,3,0,3,3) |
| 2   | (0,0,2,2,1,-1,0,0,3310,3) | (4,4,1,0,1,1) | (4,8,2,0,2,2) | (4,12,3,0,3,3) |

(2 rows)

See Also

[ST\\_AsGDALRaster](#)

10.4 Raster Accessors

10.4.1 ST\_GeoReference

**ST\_GeoReference** — Returns the georeference meta data in GDAL or ESRI format as commonly seen in a world file. Default is GDAL.

Synopsis

text **ST\_GeoReference**(raster rast, text format=GDAL);

---

Description

Returns the georeference meta data including carriage return in GDAL or ESRI format as commonly seen in a [world file](#). Default is GDAL if no type specified. type is string 'GDAL' or 'ESRI'.

Difference between format representations is as follows:

GDAL:

```
scalex
skewy
skewx
scaley
upperleftx
upperlefty
```

ESRI:

```
scalex
skewy
skewx
scaley
upperleftx + scalex*0.5
upperlefty + scaley*0.5
```

Examples

```
SELECT ST_GeoReference(rast, 'ESRI') As esri_ref, ST_GeoReference(rast, 'GDAL') As gdal_ref
FROM dummy_rast WHERE rid=1;
```

| esri_ref     | gdal_ref     |
|--------------|--------------|
| 2.0000000000 | 2.0000000000 |
| 0.0000000000 | 0.0000000000 |
| 0.0000000000 | 0.0000000000 |
| 3.0000000000 | 3.0000000000 |
| 1.5000000000 | 0.5000000000 |
| 2.0000000000 | 0.5000000000 |

See Also

[ST\\_SetGeoReference](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#)

10.4.2 ST\_Height

ST\_Height — Returns the height of the raster in pixels.

Synopsis

integer **ST\_Height**(raster rast);

Description

Returns the height of the raster.

Examples

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;
```

| rid | rastheight |
|-----|------------|
| 1   | 20         |
| 2   | 5          |

See Also

[ST\\_Width](#)

10.4.3 ST\_IsEmpty

ST\_IsEmpty — Returns true if the raster is empty (width = 0 and height = 0). Otherwise, returns false.

Synopsis

```
boolean ST_IsEmpty(raster rast);
```

Description

Returns true if the raster is empty (width = 0 and height = 0). Otherwise, returns false.

Availability: 2.0.0

Examples

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0))
st_isempty |
-----+
f |

SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0))
st_isempty |
-----+
t |
```

See Also

[ST\\_HasNoBand](#)

10.4.4 ST\_MemSize

ST\_MemSize — Returns the amount of space (in bytes) the raster takes.

Synopsis

```
integer ST_MemSize(raster rast);
```

---

Description

Returns the amount of space (in bytes) the raster takes.

This is a nice compliment to PostgreSQL built in functions `pg_column_size`, `pg_size_pretty`, `pg_relation_size`, `pg_total_relation_size`.



**Note**

`pg_relation_size` which gives the byte size of a table may return byte size lower than `ST_MemSize`. This is because `pg_relation_size` does not add toasted table contribution and large geometries are stored in TOAST tables. `pg_column_size` might return lower because it returns the compressed size. `pg_total_relation_size` - includes, the table, the toasted tables, and the indexes.

Availability: 2.2.0

Examples

```
SELECT ST_MemSize(ST_AsRaster(ST_Buffer(ST_Point(1,5),10,1000),150, 150, '8BUI')) As rast_mem;

rast_mem

22568
```

See Also

10.4.5 ST\_MetaData

`ST_MetaData` — Returns basic meta data about a raster object such as pixel size, rotation (skew), upper, lower left, etc.

Synopsis

record `ST_MetaData`(raster rast);

Description

Returns basic meta data about a raster object such as pixel size, rotation (skew), upper, lower left, etc. Columns returned: `upperleftx` | `upperlefty` | `width` | `height` | `scalex` | `scaley` | `skewx` | `skewy` | `srid` | `numbands`

Examples

```
SELECT rid, (foo.md).*
FROM (SELECT rid, ST_MetaData(rast) As md
FROM dummy_rast) As foo;
```

| rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands |
|-----|------------|------------|-------|--------|--------|--------|-------|-------|------|----------|
| 1   | 0.5        | 0.5        | 10    | 20     | 2      | 3      | 0     | 0     | 0    | 0        |
| 2   | 3427927.75 | 5793244    | 5     | 5      | 0.05   | -0.05  | 0     | 0     | 0    | 3        |

**See Also**

[ST\\_BandMetaData](#), [ST\\_NumBands](#)

**10.4.6 ST\_NumBands**

**ST\_NumBands** — Returns the number of bands in the raster object.

**Synopsis**

integer **ST\_NumBands**(raster rast);

**Description**

Returns the number of bands in the raster object.

**Examples**

```
SELECT rid, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

| rid | numbands |
|-----|----------|
| 1   | 0        |
| 2   | 3        |

**See Also**

[ST\\_Value](#)

**10.4.7 ST\_PixelHeight**

**ST\_PixelHeight** — Returns the pixel height in geometric units of the spatial reference system.

**Synopsis**

double precision **ST\_PixelHeight**(raster rast);

**Description**

Returns the height of a pixel in geometric units of the spatial reference system. In the common case where there is no skew, the pixel height is just the scale ratio between geometric coordinates and raster pixels.

Refer to [ST\\_PixelWidth](#) for a diagrammatic visualization of the relationship.

---

Examples: Rasters with no skew

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
 ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
 ST_SkewY(rast) As skewy
FROM dummy_rast;
```

| rastheight | pixheight | scalex | scaley | skewx | skewy |
|------------|-----------|--------|--------|-------|-------|
| 20         | 3         | 2      | 3      | 0     | 0     |
| 5          | 0.05      | 0.05   | -0.05  | 0     | 0     |

Examples: Rasters with skew different than 0

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
 ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
 ST_SkewY(rast) As skewy
FROM (SELECT ST_SetSkew(rast,0.5,0.5) As rast
 FROM dummy_rast) As skewed;
```

| rastheight | pixheight         | scalex | scaley | skewx | skewy |
|------------|-------------------|--------|--------|-------|-------|
| 20         | 3.04138126514911  | 2      | 3      | 0.5   | 0.5   |
| 5          | 0.502493781056044 | 0.05   | -0.05  | 0.5   | 0.5   |

See Also

[ST\\_PixelWidth](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_SkewX](#), [ST\\_SkewY](#)

10.4.8 ST\_PixelWidth

ST\_PixelWidth — Returns the pixel width in geometric units of the spatial reference system.

Synopsis

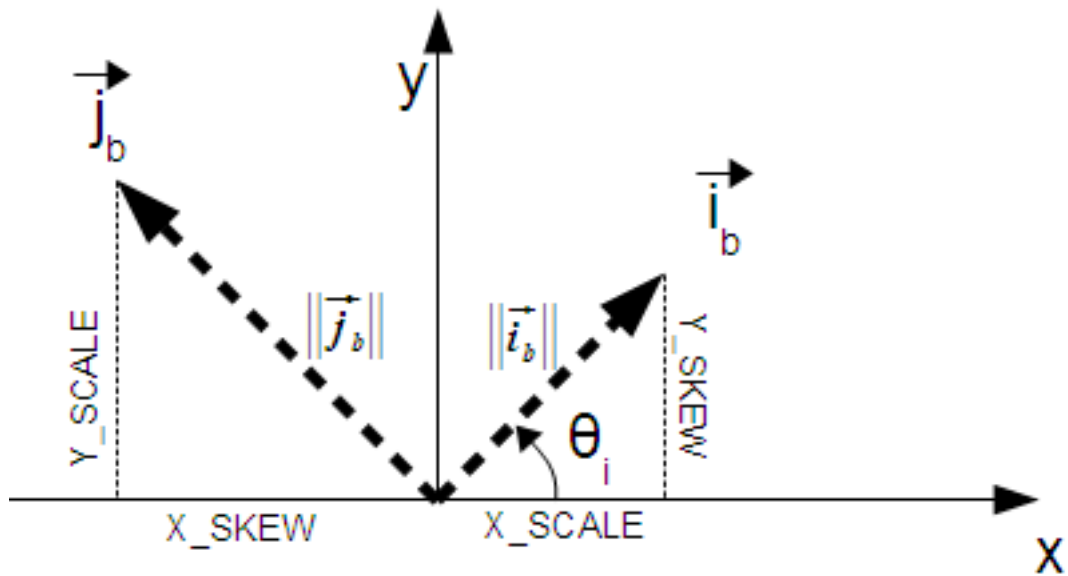
double precision **ST\_PixelWidth**(raster rast);

Description

Returns the width of a pixel in geometric units of the spatial reference system. In the common case where there is no skew, the pixel width is just the scale ratio between geometric coordinates and raster pixels.

The following diagram demonstrates the relationship:





Pixel Width: Pixel size in the *i* direction  
Pixel Height: Pixel size in the *j* direction

Examples: Rasters with no skew

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
 ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
 ST_SkewY(rast) As skewy
FROM dummy_rast;
```

| rastwidth | pixwidth | scalex | scaley | skewx | skewy |
|-----------|----------|--------|--------|-------|-------|
| 10        | 2        | 2      | 3      | 0     | 0     |
| 5         | 0.05     | 0.05   | -0.05  | 0     | 0     |

Examples: Rasters with skew different than 0

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
 ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
 ST_SkewY(rast) As skewy
FROM (SELECT ST_SetSkew(rast,0.5,0.5) As rast
FROM dummy_rast) As skewed;
```

| rastwidth | pixwidth          | scalex | scaley | skewx | skewy |
|-----------|-------------------|--------|--------|-------|-------|
| 10        | 2.06155281280883  | 2      | 3      | 0.5   | 0.5   |
| 5         | 0.502493781056044 | 0.05   | -0.05  | 0.5   | 0.5   |

See Also

[ST\\_PixelHeight](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_SkewX](#), [ST\\_SkewY](#)

10.4.9 ST\_ScaleX

ST\_ScaleX — Returns the X component of the pixel width in units of coordinate reference system.

### Synopsis

float8 **ST\_ScaleX**(raster rast);

### Description

Returns the X component of the pixel width in units of coordinate reference system. Refer to [World File](#) for more details.

Changed: 2.0.0. In WKTRaster versions this was called ST\_PixelSizeX.

### Examples

```
SELECT rid, ST_ScaleX(rast) As rastpixwidth
FROM dummy_rast;
```

| rid | rastpixwidth |
|-----|--------------|
| 1   | 2            |
| 2   | 0.05         |

### See Also

[ST\\_Width](#)

## 10.4.10 ST\_ScaleY

**ST\_ScaleY** — Returns the Y component of the pixel height in units of coordinate reference system.

### Synopsis

float8 **ST\_ScaleY**(raster rast);

### Description

Returns the Y component of the pixel height in units of coordinate reference system. May be negative. Refer to [World File](#) for more details.

Changed: 2.0.0. In WKTRaster versions this was called ST\_PixelSizeY.

### Examples

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;
```

| rid | rastpixheight |
|-----|---------------|
| 1   | 3             |
| 2   | -0.05         |

### See Also

[ST\\_Height](#)

---

### 10.4.11 ST\_RasterToWorldCoord

**ST\_RasterToWorldCoord** — Returns the raster’s upper left corner as geometric X and Y (longitude and latitude) given a column and row. Column and row starts at 1.

#### Synopsis

record **ST\_RasterToWorldCoord**(raster rast, integer xcolumn, integer yrow);

#### Description

Returns the upper left corner as geometric X and Y (longitude and latitude) given a column and row. Returned X and Y are in geometric units of the georeferenced raster. Numbering of column and row starts at 1 but if either parameter is passed a zero, a negative number or a number greater than the respective dimension of the raster, it will return coordinates outside of the raster assuming the raster’s grid is applicable outside the raster’s bounds.

Availability: 2.1.0

#### Examples

```
-- non-skewed raster
SELECT
 rid,
 (ST_RasterToWorldCoord(rast,1, 1)).*,
 (ST_RasterToWorldCoord(rast,2, 2)).*
FROM dummy_rast
```

| rid | longitude  | latitude | longitude | latitude   |
|-----|------------|----------|-----------|------------|
| 1   | 0.5        | 0.5      | 2.5       | 3.5        |
| 2   | 3427927.75 | 5793244  | 3427927.8 | 5793243.95 |

```
-- skewed raster
SELECT
 rid,
 (ST_RasterToWorldCoord(rast, 1, 1)).*,
 (ST_RasterToWorldCoord(rast, 2, 3)).*
FROM (
 SELECT
 rid,
 ST_SetSkew(rast, 100.5, 0) As rast
 FROM dummy_rast
) As foo
```

| rid | longitude  | latitude | longitude | latitude  |
|-----|------------|----------|-----------|-----------|
| 1   | 0.5        | 0.5      | 203.5     | 6.5       |
| 2   | 3427927.75 | 5793244  | 3428128.8 | 5793243.9 |

#### See Also

[ST\\_RasterToWorldCoordX](#), [ST\\_RasterToWorldCoordY](#), [ST\\_SetSkew](#)

### 10.4.12 ST\_RasterToWorldCoordX

**ST\_RasterToWorldCoordX** — Returns the geometric X coordinate upper left of a raster, column and row. Numbering of columns and rows starts at 1.

Synopsis

float8 **ST\_RasterToWorldCoordX**(raster rast, integer xcolumn);  
float8 **ST\_RasterToWorldCoordX**(raster rast, integer xcolumn, integer yrow);

Description

Returns the upper left X coordinate of a raster column row in geometric units of the georeferenced raster. Numbering of columns and rows starts at 1 but if you pass in a negative number or number higher than number of columns in raster, it will give you coordinates outside of the raster file to left or right with the assumption that the skew and pixel sizes are same as selected raster.



**Note**  
For non-skewed rasters, providing the X column is sufficient. For skewed rasters, the georeferenced coordinate is a function of the ST\_ScaleX and ST\_SkewX and row and column. An error will be raised if you give just the X column for a skewed raster.

Changed: 2.1.0 In prior versions, this was called ST\_Raster2WorldCoordX

Examples

```
-- non-skewed raster providing column is sufficient
SELECT rid, ST_RasterToWorldCoordX(rast,1) As x1coord,
 ST_RasterToWorldCoordX(rast,2) As x2coord,
 ST_ScaleX(rast) As pixelx
FROM dummy_rast;
```

| rid | x1coord    | x2coord   | pixelx |
|-----|------------|-----------|--------|
| 1   | 0.5        | 2.5       | 2      |
| 2   | 3427927.75 | 3427927.8 | 0.05   |

```
-- for fun lets skew it
SELECT rid, ST_RasterToWorldCoordX(rast, 1, 1) As x1coord,
 ST_RasterToWorldCoordX(rast, 2, 3) As x2coord,
 ST_ScaleX(rast) As pixelx
FROM (SELECT rid, ST_SetSkew(rast, 100.5, 0) As rast FROM dummy_rast) As foo;
```

| rid | x1coord    | x2coord   | pixelx |
|-----|------------|-----------|--------|
| 1   | 0.5        | 203.5     | 2      |
| 2   | 3427927.75 | 3428128.8 | 0.05   |

See Also

[ST\\_ScaleX](#), [ST\\_RasterToWorldCoordY](#), [ST\\_SetSkew](#), [ST\\_SkewX](#)

10.4.13 ST\_RasterToWorldCoordY

**ST\_RasterToWorldCoordY** — Returns the geometric Y coordinate upper left corner of a raster, column and row. Numbering of columns and rows starts at 1.

Synopsis

```
float8 ST_RasterToWorldCoordY(raster rast, integer yrow);
float8 ST_RasterToWorldCoordY(raster rast, integer xcolumn, integer yrow);
```

Description

Returns the upper left Y coordinate of a raster column row in geometric units of the georeferenced raster. Numbering of columns and rows starts at 1 but if you pass in a negative number or number higher than number of columns/rows in raster, it will give you coordinates outside of the raster file to left or right with the assumption that the skew and pixel sizes are same as selected raster tile.



**Note** For non-skewed rasters, providing the Y column is sufficient. For skewed rasters, the georeferenced coordinate is a function of the ST\_ScaleY and ST\_SkewY and row and column. An error will be raised if you give just the Y row for a skewed raster.

Changed: 2.1.0 In prior versions, this was called ST\_Raster2WorldCoordY

Examples

```
-- non-skewed raster providing row is sufficient
SELECT rid, ST_RasterToWorldCoordY(rast,1) As ylcoord,
 ST_RasterToWorldCoordY(rast,3) As y2coord,
 ST_ScaleY(rast) As pixely
FROM dummy_rast;

rid | ylcoord | y2coord | pixely
-----+-----+-----+-----
 1 | 0.5 | 6.5 | 3
 2 | 5793244 | 5793243.9 | -0.05

-- for fun lets skew it
SELECT rid, ST_RasterToWorldCoordY(rast,1,1) As ylcoord,
 ST_RasterToWorldCoordY(rast,2,3) As y2coord,
 ST_ScaleY(rast) As pixely
FROM (SELECT rid, ST_SetSkew(rast,0,100.5) As rast FROM dummy_rast) As foo;

rid | ylcoord | y2coord | pixely
-----+-----+-----+-----
 1 | 0.5 | 107 | 3
 2 | 5793244 | 5793344.4 | -0.05
```

See Also

[ST\\_ScaleY](#), [ST\\_RasterToWorldCoordX](#), [ST\\_SetSkew](#), [ST\\_SkewY](#)

10.4.14 ST\_Rotation

ST\_Rotation — Returns the rotation of the raster in radian.

Synopsis

```
float8 ST_Rotation(raster rast);
```

Description

Returns the uniform rotation of the raster in radian. If a raster does not have uniform rotation, NaN is returned. Refer to [World File](#) for more details.

Examples

```
SELECT rid, ST_Rotation(ST_SetScale(ST_SetSkew(rast, sqrt(2)), sqrt(2))) as rot FROM ↵
dummy_rast;

rid | rot
-----+-----
 1 | 0.785398163397448
 2 | 0.785398163397448
```

See Also

[ST\\_SetRotation](#), [ST\\_SetScale](#), [ST\\_SetSkew](#)

10.4.15 ST\_SkewX

ST\_SkewX — Returns the georeference X skew (or rotation parameter).

Synopsis

float8 **ST\_SkewX**(raster rast);

Description

Returns the georeference X skew (or rotation parameter). Refer to [World File](#) for more details.

Examples

```
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
 ST_GeoReference(rast) as georef
FROM dummy_rast;

rid | skewx | skewy | georef
-----+-----+-----+-----
 1 | 0 | 0 | 2.0000000000
 : 0.0000000000
 : 0.0000000000
 : 3.0000000000
 : 0.5000000000
 : 0.5000000000
 :
 2 | 0 | 0 | 0.0500000000
 : 0.0000000000
 : 0.0000000000
 : -0.0500000000
 : 3427927.7500000000
 : 5793244.0000000000
```

See Also

[ST\\_GeoReference](#), [ST\\_SkewY](#), [ST\\_SetSkew](#)

10.4.16 ST\_SkewY

ST\_SkewY — Returns the georeference Y skew (or rotation parameter).

Synopsis

float8 **ST\_SkewY**(raster rast);

Description

Returns the georeference Y skew (or rotation parameter). Refer to [World File](#) for more details.

Examples

```
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
 ST_GeoReference(rast) as georef
FROM dummy_rast;
```

| rid | skewx | skewy | georef               |
|-----|-------|-------|----------------------|
| 1   | 0     | 0     | 2.0000000000         |
|     |       |       | : 0.0000000000       |
|     |       |       | : 0.0000000000       |
|     |       |       | : 3.0000000000       |
|     |       |       | : 0.5000000000       |
|     |       |       | : 0.5000000000       |
|     |       |       | :                    |
| 2   | 0     | 0     | 0.0500000000         |
|     |       |       | : 0.0000000000       |
|     |       |       | : 0.0000000000       |
|     |       |       | : -0.0500000000      |
|     |       |       | : 3427927.7500000000 |
|     |       |       | : 5793244.0000000000 |

See Also

[ST\\_GeoReference](#), [ST\\_SkewX](#), [ST\\_SetSkew](#)

10.4.17 ST\_SRID

ST\_SRID — Returns the spatial reference identifier of the raster as defined in spatial\_ref\_sys table.

Synopsis

integer **ST\_SRID**(raster rast);

**Description**

Returns the spatial reference identifier of the raster object as defined in the spatial\_ref\_sys table.



**Note**

From PostGIS 2.0+ the srid of a non-georeferenced raster/geometry is 0 instead of the prior -1.

**Examples**

```
SELECT ST_SRID(rast) As srid
FROM dummy_rast WHERE rid=1;

srid

0
```

**See Also**

Section 4.5, [ST\\_SRID](#)

**10.4.18 ST\_Summary**

ST\_Summary — Returns a text summary of the contents of the raster.

**Synopsis**

text **ST\_Summary**(raster rast);

**Description**

Returns a text summary of the contents of the raster.

Availability: 2.1.0

**Examples**

```
SELECT ST_Summary(
 ST_AddBand(
 ST_AddBand(
 ST_AddBand(
 ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0)
 , 1, '8BUI', 1, 0
)
 , 2, '32BF', 0, -9999
)
 , 3, '16BSI', 0, NULL
)
);

 st_summary

```



```

Raster of 10x10 pixels has 3 bands and extent of BOX(0 -10,10 0)+
 band 1 of pixtype 8BUI is in-db with NODATA value of 0 +
 band 2 of pixtype 32BF is in-db with NODATA value of -9999 +
 band 3 of pixtype 16BSI is in-db with no NODATA value
(1 row)

```

### See Also

[ST\\_MetaData](#), [ST\\_BandMetaData](#), [ST\\_Summary](#) [ST\\_Extent](#)

## 10.4.19 ST\_UpperLeftX

**ST\_UpperLeftX** — Returns the upper left X coordinate of raster in projected spatial ref.

### Synopsis

```
float8 ST_UpperLeftX(raster rast);
```

### Description

Returns the upper left X coordinate of raster in projected spatial ref.

### Examples

```

SELECT rid, ST_UpperLeftX(rast) As ulx
FROM dummy_rast;

```

| rid | ulx        |
|-----|------------|
| 1   | 0.5        |
| 2   | 3427927.75 |

### See Also

[ST\\_UpperLeftY](#), [ST\\_GeoReference](#), [Box3D](#)

## 10.4.20 ST\_UpperLeftY

**ST\_UpperLeftY** — Returns the upper left Y coordinate of raster in projected spatial ref.

### Synopsis

```
float8 ST_UpperLeftY(raster rast);
```

### Description

Returns the upper left Y coordinate of raster in projected spatial ref.

Examples

```
SELECT rid, ST_UpperLeftY(rast) As uly
FROM dummy_rast;
```

| rid | uly     |
|-----|---------|
| 1   | 0.5     |
| 2   | 5793244 |

See Also

[ST\\_UpperLeftX](#), [ST\\_GeoReference](#), [Box3D](#)

10.4.21 ST\_Width

ST\_Width — Returns the width of the raster in pixels.

Synopsis

integer **ST\_Width**(raster rast);

Description

Returns the width of the raster in pixels.

Examples

```
SELECT ST_Width(rast) As rastwidth
FROM dummy_rast WHERE rid=1;
```

| rastwidth |
|-----------|
| 10        |

See Also

[ST\\_Height](#)

10.4.22 ST\_WorldToRasterCoord

ST\_WorldToRasterCoord — Returns the upper left corner as column and row given geometric X and Y (longitude and latitude) or a point geometry expressed in the spatial reference coordinate system of the raster.

Synopsis

record **ST\_WorldToRasterCoord**(raster rast, geometry pt);  
record **ST\_WorldToRasterCoord**(raster rast, double precision longitude, double precision latitude);

Description

Returns the upper left corner as column and row given geometric X and Y (longitude and latitude) or a point geometry. This function works regardless of whether or not the geometric X and Y or point geometry is outside the extent of the raster. Geometric X and Y must be expressed in the spatial reference coordinate system of the raster.

Availability: 2.1.0

Examples

```
SELECT
 rid,
 (ST_WorldToRasterCoord(rast, 3427927.8, 20.5)).*,
 (ST_WorldToRasterCoord(rast, ST_GeomFromText('POINT(3427927.8 20.5)', ST_SRID(rast)))).*
FROM dummy_rast;
```

| rid | columnx | rowy      | columnx | rowy      |
|-----|---------|-----------|---------|-----------|
| 1   | 1713964 | 7         | 1713964 | 7         |
| 2   | 2       | 115864471 | 2       | 115864471 |

See Also

[ST\\_WorldToRasterCoordX](#), [ST\\_WorldToRasterCoordY](#), [ST\\_RasterToWorldCoordX](#), [ST\\_RasterToWorldCoordY](#), [ST\\_SRID](#)

10.4.23 ST\_WorldToRasterCoordX

ST\_WorldToRasterCoordX — Returns the column in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw) represented in world spatial reference system of raster.

Synopsis

integer **ST\_WorldToRasterCoordX**(raster rast, geometry pt);  
integer **ST\_WorldToRasterCoordX**(raster rast, double precision xw);  
integer **ST\_WorldToRasterCoordX**(raster rast, double precision xw, double precision yw);

Description

Returns the column in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw). A point, or (both xw and yw world coordinates are required if a raster is skewed). If a raster is not skewed then xw is sufficient. World coordinates are in the spatial reference coordinate system of the raster.

Changed: 2.1.0 In prior versions, this was called ST\_World2RasterCoordX

Examples

```
SELECT rid, ST_WorldToRasterCoordX(rast, 3427927.8) As xcoord,
 ST_WorldToRasterCoordX(rast, 3427927.8, 20.5) As xcoord_xwyw,
 ST_WorldToRasterCoordX(rast, ST_GeomFromText('POINT(3427927.8 20.5)', ST_SRID(rast))) ←
 As ptxcoord
FROM dummy_rast;
```

| rid | xcoord  | xcoord_xwyw | ptxcoord |
|-----|---------|-------------|----------|
| 1   | 1713964 | 1713964     | 1713964  |
| 2   | 1       | 1           | 1        |

See Also

[ST\\_RasterToWorldCoordX](#), [ST\\_RasterToWorldCoordY](#), [ST\\_SRID](#)

10.4.24 ST\_WorldToRasterCoordY

ST\_WorldToRasterCoordY — Returns the row in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw) represented in world spatial reference system of raster.

Synopsis

integer **ST\_WorldToRasterCoordY**(raster rast, geometry pt);  
integer **ST\_WorldToRasterCoordY**(raster rast, double precision xw);  
integer **ST\_WorldToRasterCoordY**(raster rast, double precision xw, double precision yw);

Description

Returns the row in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw). A point, or (both xw and yw world coordinates are required if a raster is skewed). If a raster is not skewed then xw is sufficient. World coordinates are in the spatial reference coordinate system of the raster.

Changed: 2.1.0 In prior versions, this was called ST\_World2RasterCoordY

Examples

```
SELECT rid, ST_WorldToRasterCoordY(rast,20.5) As ycoord,
 ST_WorldToRasterCoordY(rast,3427927.8,20.5) As ycoord_xwyw,
 ST_WorldToRasterCoordY(rast,ST_GeomFromText('POINT(3427927.8 20.5)',ST_SRID(rast))) As ptycoord
FROM dummy_rast;
```

| rid | ycoord    | ycoord_xwyw | ptycoord  |
|-----|-----------|-------------|-----------|
| 1   | 7         | 7           | 7         |
| 2   | 115864471 | 115864471   | 115864471 |

See Also

[ST\\_RasterToWorldCoordX](#), [ST\\_RasterToWorldCoordY](#), [ST\\_SRID](#)

10.5 Raster Band Accessors

10.5.1 ST\_BandMetaData

ST\_BandMetaData — Returns basic meta data for a specific raster band. band num 1 is assumed if none-specified.

Synopsis

(1) record **ST\_BandMetaData**(raster rast, integer band=1);  
(2) record **ST\_BandMetaData**(raster rast, integer[] band);

---

Description

Returns basic meta data about a raster band. Columns returned: pixeltype, nodatavalue, isoutdb, path, outdbbandnum, filesize, filetimestamp.



**Note**  
If raster contains no bands then an error is thrown.



**Note**  
If band has no NODATA value, nodatavalue are NULL.



**Note**  
If isoutdb is False, path, outdbbandnum, filesize and filetimestamp are NULL. If outdb access is disabled, filesize and filetimestamp will also be NULL.

Enhanced: 2.5.0 to include *outdbbandnum*, *filesize* and *filetimestamp* for outdb rasters.

Examples: Variant 1

```
SELECT
 rid,
 (foo.md).*
FROM (
 SELECT
 rid,
 ST_BandMetaData(rast, 1) AS md
 FROM dummy_rast
 WHERE rid=2
) As foo;
```

| rid | pixeltype | nodatavalue | isoutdb | path | outdbbandnum |
|-----|-----------|-------------|---------|------|--------------|
| 2   | 8BUI      |             | 0       | f    |              |

Examples: Variant 2

```
WITH foo AS (
 SELECT
 ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↵
 loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
 *
FROM ST_BandMetadata(
 (SELECT rast FROM foo),
 ARRAY[1,3,2]::int[]
);
```

| bandnum | pixeltype                     | nodatavalue | isoutdb      | path                                         | ↔             |
|---------|-------------------------------|-------------|--------------|----------------------------------------------|---------------|
|         |                               |             | outdbbandnum | filesize                                     | filetimestamp |
| 1       | 8BUI                          |             | t            | /home/pele/devel/geo/postgis-git/raster/test | ↔             |
|         | /regress/loader/Projected.tif |             | 1            | 12345                                        | 1521807257    |
| 3       | 8BUI                          |             | t            | /home/pele/devel/geo/postgis-git/raster/test | ↔             |
|         | /regress/loader/Projected.tif |             | 3            | 12345                                        | 1521807257    |
| 2       | 8BUI                          |             | t            | /home/pele/devel/geo/postgis-git/raster/test | ↔             |
|         | /regress/loader/Projected.tif |             | 2            | 12345                                        | 1521807257    |

See Also

[ST\\_MetaData](#), [ST\\_BandPixelType](#)

10.5.2 ST\_BandNoDataValue

ST\_BandNoDataValue — Returns the value in a given band that represents no data. If no band num 1 is assumed.

Synopsis

double precision **ST\_BandNoDataValue**(raster rast, integer bandnum=1);

Description

Returns the value that represents no data for the band

Examples

```
SELECT ST_BandNoDataValue(rast,1) As bnval1,
 ST_BandNoDataValue(rast,2) As bnval2, ST_BandNoDataValue(rast,3) As bnval3
FROM dummy_rast
WHERE rid = 2;
```

| bnval1 | bnval2 | bnval3 |
|--------|--------|--------|
| 0      | 0      | 0      |

See Also

[ST\\_NumBands](#)

10.5.3 ST\_BandIsNoData

ST\_BandIsNoData — Returns true if the band is filled with only nodata values.

Synopsis

boolean **ST\_BandIsNoData**(raster rast, integer band, boolean forceChecking=true);  
boolean **ST\_BandIsNoData**(raster rast, boolean forceChecking=true);

## Description

Returns true if the band is filled with only nodata values. Band 1 is assumed if not specified. If the last argument is TRUE, the entire band is checked pixel by pixel. Otherwise, the function simply returns the value of the isnodata flag for the band. The default value for this parameter is FALSE, if not specified.

Availability: 2.0.0



### Note

If the flag is dirty (this is, the result is different using TRUE as last parameter and not using it) you should update the raster to set this flag to true, by using `ST_SetBandIsNodata()`, or `ST_SetBandNodataValue()` with TRUE as last argument. See [ST\\_SetBandIsNodata](#).

## Examples

```
-- Create dummy table with one raster column
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ←
 = 3.
-- In the second band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'6' -- hasnodatavalue and isnodata value set to true.
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
)
```

```
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true
select st_bandisnodata(rast, 2) from dummy_rast where rid = 1; -- Expected false
```

### See Also

[ST\\_BandNoDataValue](#), [ST\\_NumBands](#), [ST\\_SetBandNoDataValue](#), [ST\\_SetBandIsNoData](#)

## 10.5.4 ST\_BandPath

**ST\_BandPath** — Returns system file path to a band stored in file system. If no bandnum specified, 1 is assumed.

### Synopsis

text **ST\_BandPath**(raster rast, integer bandnum=1);

### Description

Returns system file path to a band. Throws an error if called with an in db band.

### Examples

### See Also

## 10.5.5 ST\_BandFileSize

**ST\_BandFileSize** — Returns the file size of a band stored in file system. If no bandnum specified, 1 is assumed.

### Synopsis

bigint **ST\_BandFileSize**(raster rast, integer bandnum=1);

### Description

Returns the file size of a band stored in file system. Throws an error if called with an in db band, or if outdb access is not enabled.

This function is typically used in conjunction with **ST\_BandPath**() and **ST\_BandFileTimestamp**() so a client can determine if the filename of a outdb raster as seen by it is the same as the one seen by the server.

Availability: 2.5.0

---



## Examples

```
SELECT ST_BandFileSize(rast,1) FROM dummy_rast WHERE rid = 1;

 st_bandfilesize

 240574
```

### 10.5.6 ST\_BandFileTimestamp

**ST\_BandFileTimestamp** — Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.

#### Synopsis

**bigint** **ST\_BandFileTimestamp**(raster rast, integer bandnum=1);

#### Description

Returns the file timestamp (number of seconds since Jan 1st 1970 00:00:00 UTC) of a band stored in file system. Throws an error if called with an in db band, or if outdb access is not enabled.

This function is typically used in conjunction with **ST\_BandPath()** and **ST\_BandFileSize()** so a client can determine if the filename of a outdb raster as seen by it is the same as the one seen by the server.

Availability: 2.5.0

## Examples

```
SELECT ST_BandFileTimestamp(rast,1) FROM dummy_rast WHERE rid = 1;

 st_bandfiletimestamp

 1521807257
```

### 10.5.7 ST\_BandPixelType

**ST\_BandPixelType** — Returns the type of pixel for given band. If no bandnum specified, 1 is assumed.

#### Synopsis

**text** **ST\_BandPixelType**(raster rast, integer bandnum=1);

#### Description

Returns name describing data type and size of values stored in each cell of given band.

There are 11 pixel types. Pixel Types supported are as follows:

- 1BB - 1-bit boolean
- 2BUI - 2-bit unsigned integer
- 4BUI - 4-bit unsigned integer

- 8BSI - 8-bit signed integer
- 8BUI - 8-bit unsigned integer
- 16BSI - 16-bit signed integer
- 16BUI - 16-bit unsigned integer
- 32BSI - 32-bit signed integer
- 32BUI - 32-bit unsigned integer
- 32BF - 32-bit float
- 64BF - 64-bit float

**Examples**

```
SELECT ST_BandPixelType(rast,1) As btype1,
 ST_BandPixelType(rast,2) As btype2, ST_BandPixelType(rast,3) As btype3
FROM dummy_rast
WHERE rid = 2;
```

| btype1 | btype2 | btype3 |
|--------|--------|--------|
| 8BUI   | 8BUI   | 8BUI   |

**See Also**

[ST\\_NumBands](#)

**10.5.8 ST\_MinPossibleValue**

ST\_MinPossibleValue — Returns the minimum value this pixeltype can store.

**Synopsis**

integer **ST\_MinPossibleValue**(text pixeltype);

**Description**

Returns the minimum value this pixeltype can store.

**Examples**

```
SELECT ST_MinPossibleValue('16BSI');
```

| st_minpossiblevalue |
|---------------------|
| -32768              |

```
SELECT ST_MinPossibleValue('8BUI');
```

| st_minpossiblevalue |
|---------------------|
| 0                   |

See Also

[ST\\_BandPixelType](#)

10.5.9 ST\_HasNoBand

ST\_HasNoBand — Returns true if there is no band with given band number. If no band number is specified, then band number 1 is assumed.

Synopsis

boolean **ST\_HasNoBand**(raster rast, integer bandnum=1);

Description

Returns true if there is no band with given band number. If no band number is specified, then band number 1 is assumed.

Availability: 2.0.0

Examples

```
SELECT rid, ST_HasNoBand(rast) As hb1, ST_HasNoBand(rast,2) as hb2,
ST_HasNoBand(rast,4) as hb4, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

| rid | hb1 | hb2 | hb4 | numbands |
|-----|-----|-----|-----|----------|
| 1   | t   | t   | t   | 0        |
| 2   | f   | f   | t   | 3        |

See Also

[ST\\_NumBands](#)

10.6 Raster Pixel Accessors and Setters

10.6.1 ST\_PixelAsPolygon

ST\_PixelAsPolygon — Returns the polygon geometry that bounds the pixel for a particular row and column.

Synopsis

geometry **ST\_PixelAsPolygon**(raster rast, integer columnx, integer rowy);

Description

Returns the polygon geometry that bounds the pixel for a particular row and column.

Availability: 2.0.0

---

Examples

```
-- get raster pixel polygon
SELECT i,j, ST_AsText(ST_PixelAsPolygon(foo.rast, i,j)) As blpgeom
FROM dummy_rast As foo
 CROSS JOIN generate_series(1,2) As i
 CROSS JOIN generate_series(1,1) As j
WHERE rid=2;
```

| i | j | blpgeom                                                                 |
|---|---|-------------------------------------------------------------------------|
| 1 | 1 | POLYGON((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...  |
| 2 | 1 | POLYGON((3427927.8 5793244,3427927.85 5793244,3427927.85 5793243.95, .. |

See Also

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroid](#), [ST\\_PixelAsCentroids](#), [ST\\_Intersection](#), [ST\\_AsText](#)

10.6.2 ST\_PixelAsPolygons

ST\_PixelAsPolygons — Returns the polygon geometry that bounds every pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel.

Synopsis

setof record **ST\_PixelAsPolygons**(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);

Description

Returns the polygon geometry that bounds every pixel of a raster band along with the value (double precision), the X and the Y raster coordinates (integers) of each pixel.

Return record format: *geom* **geometry**, *val* double precision, *x* integer, *y* integers.



**Note**  
When *exclude\_nodata\_value* = TRUE, only those pixels whose values are not NODATA are returned as points.



**Note**  
ST\_PixelAsPolygons returns one polygon geometry for every pixel. This is different than ST\_DumpAsPolygons where each geometry represents one or more pixels with the same pixel value.

- Availability: 2.0.0
- Enhanced: 2.1.0 *exclude\_nodata\_value* optional argument was added.
- Changed: 2.1.1 Changed behavior of *exclude\_nodata\_value*.

Examples

```
-- get raster pixel polygon
SELECT (gv).x, (gv).y, (gv).val, ST_AsText((gv).geom) geom
FROM (SELECT ST_PixelAsPolygons(
 ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.001, 0.001, 0.001, 4269),
 '8BUI'::text, 1, 0),
 2, 2, 10),
 1, 1, NULL)
) gv
) foo;
```

| x | y | val | geom                                                                   |
|---|---|-----|------------------------------------------------------------------------|
| 1 | 1 |     | POLYGON((0 0,0.001 0.001,0.002 0,0.001 -0.001,0 0))                    |
| 1 | 2 | 1   | POLYGON((0.001 -0.001,0.002 0,0.003 -0.001,0.002 -0.002,0.001 -0.001)) |
| 2 | 1 | 1   | POLYGON((0.001 0.001,0.002 0.002,0.003 0.001,0.002 0,0.001 0.001))     |
| 2 | 2 | 10  | POLYGON((0.002 0,0.003 0.001,0.004 0,0.003 -0.001,0.002 0))            |

See Also

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroid](#), [ST\\_PixelAsCentroids](#), [ST\\_AsText](#)

10.6.3 ST\_PixelAsPoint

ST\_PixelAsPoint — Returns a point geometry of the pixel’s upper-left corner.

Synopsis

geometry **ST\_PixelAsPoint**(raster rast, integer columnx, integer rowy);

Description

Returns a point geometry of the pixel’s upper-left corner.

Availability: 2.1.0

Examples

```
SELECT ST_AsText(ST_PixelAsPoint(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;
```

| st_astext      |
|----------------|
| POINT(0.5 0.5) |

See Also

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroid](#), [ST\\_PixelAsCentroids](#)

10.6.4 ST\_PixelAsPoints

ST\_PixelAsPoints — Returns a point geometry for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The coordinates of the point geometry are of the pixel’s upper-left corner.

Synopsis

setof record ST\_PixelAsPoints(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);

Description

Returns a point geometry for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The coordinates of the point geometry are of the pixel’s upper-left corner.

Return record format: *geom* **geometry**, *val* double precision, *x* integer, *y* integers.



**Note**  
When *exclude\_nodata\_value* = TRUE, only those pixels whose values are not NODATA are returned as points.

Availability: 2.1.0  
Changed: 2.1.1 Changed behavior of *exclude\_nodata\_value*.

Examples

```
SELECT x, y, val, ST_AsText(geom) FROM (SELECT (ST_PixelAsPoints(rast, 1)).* FROM ↵
 dummy_rast WHERE rid = 2) foo;
```

| x | y | val | st_astext                    |
|---|---|-----|------------------------------|
| 1 | 1 | 253 | POINT(3427927.75 5793244)    |
| 2 | 1 | 254 | POINT(3427927.8 5793244)     |
| 3 | 1 | 253 | POINT(3427927.85 5793244)    |
| 4 | 1 | 254 | POINT(3427927.9 5793244)     |
| 5 | 1 | 254 | POINT(3427927.95 5793244)    |
| 1 | 2 | 253 | POINT(3427927.75 5793243.95) |
| 2 | 2 | 254 | POINT(3427927.8 5793243.95)  |
| 3 | 2 | 254 | POINT(3427927.85 5793243.95) |
| 4 | 2 | 253 | POINT(3427927.9 5793243.95)  |
| 5 | 2 | 249 | POINT(3427927.95 5793243.95) |
| 1 | 3 | 250 | POINT(3427927.75 5793243.9)  |
| 2 | 3 | 254 | POINT(3427927.8 5793243.9)   |
| 3 | 3 | 254 | POINT(3427927.85 5793243.9)  |
| 4 | 3 | 252 | POINT(3427927.9 5793243.9)   |
| 5 | 3 | 249 | POINT(3427927.95 5793243.9)  |
| 1 | 4 | 251 | POINT(3427927.75 5793243.85) |
| 2 | 4 | 253 | POINT(3427927.8 5793243.85)  |
| 3 | 4 | 254 | POINT(3427927.85 5793243.85) |
| 4 | 4 | 254 | POINT(3427927.9 5793243.85)  |
| 5 | 4 | 253 | POINT(3427927.95 5793243.85) |
| 1 | 5 | 252 | POINT(3427927.75 5793243.8)  |
| 2 | 5 | 250 | POINT(3427927.8 5793243.8)   |
| 3 | 5 | 254 | POINT(3427927.85 5793243.8)  |
| 4 | 5 | 254 | POINT(3427927.9 5793243.8)   |
| 5 | 5 | 254 | POINT(3427927.95 5793243.8)  |

**See Also**

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsCentroid](#), [ST\\_PixelAsCentroids](#)

**10.6.5 ST\_PixelAsCentroid**

**ST\_PixelAsCentroid** — Returns the centroid (point geometry) of the area represented by a pixel.

**Synopsis**

geometry **ST\_PixelAsCentroid**(raster rast, integer x, integer y);

**Description**

Returns the centroid (point geometry) of the area represented by a pixel.

Enhanced: 3.2.0 Faster now implemented in C.

Availability: 2.1.0

**Examples**

```
SELECT ST_AsText(ST_PixelAsCentroid(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;

 st_astext

POINT(1.5 2)
```

**See Also**

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroids](#)

**10.6.6 ST\_PixelAsCentroids**

**ST\_PixelAsCentroids** — Returns the centroid (point geometry) for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The point geometry is the centroid of the area represented by a pixel.

**Synopsis**

setof record **ST\_PixelAsCentroids**(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);

**Description**

Returns the centroid (point geometry) for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The point geometry is the centroid of the area represented by a pixel.

Return record format: *geom* **geometry**, *val* double precision, *x* integer, *y* integers.

**Note**

When `exclude_nodata_value = TRUE`, only those pixels whose values are not NODATA are returned as points.

Enhanced: 3.2.0 Faster now implemented in C.

Changed: 2.1.1 Changed behavior of `exclude_nodata_value`.

Availability: 2.1.0

Examples

```
--LATERAL syntax requires PostgreSQL 9.3+
SELECT x, y, val, ST_AsText(geom)
 FROM (SELECT dp.* FROM dummy_rast, LATERAL ST_PixelAsCentroids(rast, 1) AS dp WHERE rid <=
 = 2) foo;
 x | y | val | st_astext
---+---+-----+-----
 1 | 1 | 253 | POINT(3427927.775 5793243.975)
 2 | 1 | 254 | POINT(3427927.825 5793243.975)
 3 | 1 | 253 | POINT(3427927.875 5793243.975)
 4 | 1 | 254 | POINT(3427927.925 5793243.975)
 5 | 1 | 254 | POINT(3427927.975 5793243.975)
 1 | 2 | 253 | POINT(3427927.775 5793243.925)
 2 | 2 | 254 | POINT(3427927.825 5793243.925)
 3 | 2 | 254 | POINT(3427927.875 5793243.925)
 4 | 2 | 253 | POINT(3427927.925 5793243.925)
 5 | 2 | 249 | POINT(3427927.975 5793243.925)
 1 | 3 | 250 | POINT(3427927.775 5793243.875)
 2 | 3 | 254 | POINT(3427927.825 5793243.875)
 3 | 3 | 254 | POINT(3427927.875 5793243.875)
 4 | 3 | 252 | POINT(3427927.925 5793243.875)
 5 | 3 | 249 | POINT(3427927.975 5793243.875)
 1 | 4 | 251 | POINT(3427927.775 5793243.825)
 2 | 4 | 253 | POINT(3427927.825 5793243.825)
 3 | 4 | 254 | POINT(3427927.875 5793243.825)
 4 | 4 | 254 | POINT(3427927.925 5793243.825)
 5 | 4 | 253 | POINT(3427927.975 5793243.825)
 1 | 5 | 252 | POINT(3427927.775 5793243.775)
 2 | 5 | 250 | POINT(3427927.825 5793243.775)
 3 | 5 | 254 | POINT(3427927.875 5793243.775)
 4 | 5 | 254 | POINT(3427927.925 5793243.775)
 5 | 5 | 254 | POINT(3427927.975 5793243.775)
```

See Also

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroid](#)

10.6.7 ST\_Value

**ST\_Value** — Returns the value of a given band in a given columnx, rowy pixel or at a particular geometric point. Band numbers start at 1 and assumed to be 1 if not specified. If `exclude_nodata_value` is set to false, then all pixels include nodata pixels are considered to intersect and return value. If `exclude_nodata_value` is not passed in then reads it from metadata of raster.

Synopsis

```
double precision ST_Value(raster rast, geometry pt, boolean exclude_nodata_value=true);
double precision ST_Value(raster rast, integer band, geometry pt, boolean exclude_nodata_value=true, text resample='nearest');
double precision ST_Value(raster rast, integer x, integer y, boolean exclude_nodata_value=true);
double precision ST_Value(raster rast, integer band, integer x, integer y, boolean exclude_nodata_value=true);
```



## Description

Returns the value of a given band in a given columnx, rowy pixel or at a given geometry point. Band numbers start at 1 and band is assumed to be 1 if not specified.

If `exclude_nodata_value` is set to `true`, then only non nodata pixels are considered. If `exclude_nodata_value` is set to `false`, then all pixels are considered.

The allowed values of the `resample` parameter are "nearest" which performs the default nearest-neighbor resampling, and "bilinear" which performs a **bilinear interpolation** to estimate the value between pixel centers.

Enhanced: 3.2.0 `resample` optional argument was added.

Enhanced: 2.0.0 `exclude_nodata_value` optional argument was added.

## Examples

```
-- get raster values at particular postgis geometry points
-- the srid of your geometry should be same as for your raster
SELECT rid, ST_Value(rast, foo.pt_geom) As b1pval, ST_Value(rast, 2, foo.pt_geom) As b2pval
FROM dummy_rast CROSS JOIN (SELECT ST_SetSRID(ST_Point(3427927.77, 5793243.76), 0) As
 pt_geom) As foo
WHERE rid=2;
```

| rid | b1pval | b2pval |
|-----|--------|--------|
| 2   | 252    | 79     |

```
-- general fictitious example using a real table
SELECT rid, ST_Value(rast, 3, sometable.geom) As b3pval
FROM sometable
WHERE ST_Intersects(rast, sometable.geom);
```

```
SELECT rid, ST_Value(rast, 1, 1, 1) As b1pval,
 ST_Value(rast, 2, 1, 1) As b2pval, ST_Value(rast, 3, 1, 1) As b3pval
FROM dummy_rast
WHERE rid=2;
```

| rid | b1pval | b2pval | b3pval |
|-----|--------|--------|--------|
| 2   | 253    | 78     | 70     |

```
--- Get all values in bands 1,2,3 of each pixel ---
SELECT x, y, ST_Value(rast, 1, x, y) As b1val,
 ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1, 1000) As x CROSS JOIN generate_series(1, 1000) As y
WHERE rid = 2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);
```

| x | y | b1val | b2val | b3val |
|---|---|-------|-------|-------|
| 1 | 1 | 253   | 78    | 70    |
| 1 | 2 | 253   | 96    | 80    |
| 1 | 3 | 250   | 99    | 90    |
| 1 | 4 | 251   | 89    | 77    |
| 1 | 5 | 252   | 79    | 62    |
| 2 | 1 | 254   | 98    | 86    |
| 2 | 2 | 254   | 118   | 108   |
| : |   |       |       |       |
| : |   |       |       |       |

```

--- Get all values in bands 1,2,3 of each pixel same as above but returning the upper left ←
point point of each pixel --
SELECT ST_AsText(ST_SetSRID(
 ST_Point(ST_UpperLeftX(rast) + ST_ScaleX(rast)*x,
 ST_UpperLeftY(rast) + ST_ScaleY(rast)*y),
 ST_SRID(rast))) As uplpt
 , ST_Value(rast, 1, x, y) As b1val,
 ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid = 2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);

```

| uplpt                       | b1val   b2val   b3val |
|-----------------------------|-----------------------|
| POINT(3427929.25 5793245.5) | 253   78   70         |
| POINT(3427929.25 5793247)   | 253   96   80         |
| POINT(3427929.25 5793248.5) | 250   99   90         |

```

:
```

```

--- Get a polygon formed by union of all pixels
that fall in a particular value range and intersect particular polygon --
SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
 ST_UpperLeftX(rast), ST_UpperLeftY(rast),
 ST_UpperLeftX(rast) + ST_ScaleX(rast),
 ST_UpperLeftY(rast) + ST_ScaleY(rast), 0
), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid = 2
AND x <= ST_Width(rast) AND y <= ST_Height(rast)) As foo
WHERE
 ST_Intersects(
 pixpolyg,
 ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928 ←
 5793243.75,3427928 5793244))',0)
) AND b2val != 254;

```

| shadow                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MULTIPOLYGON(((3427928 5793243.9,3427928 5793243.85,3427927.95 5793243.85,3427927.95 ←<br>5793243.9,<br>3427927.95 5793243.95,3427928 5793243.95,3427928.05 5793243.95,3427928.05 ←<br>5793243.9,3427928 5793243.9)),((3427927.95 5793243.9,3427927.95 579324<br>3.85,3427927.9 5793243.85,3427927.85 5793243.85,3427927.85 5793243.9,3427927.9 ←<br>5793243.9,3427927.9 5793243.95,<br>3427927.95 5793243.95,3427927.95 5793243.9)),((3427927.85 5793243.75,3427927.85 ←<br>5793243.7,3427927.8 5793243.7,3427927.8 5793243.75<br>,3427927.8 5793243.8,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85 ←<br>5793243.8,3427927.85 5793243.75)),<br>((3427928.05 5793243.75,3427928.05 5793243.7,3427928 5793243.7,3427927.95 ←<br>5793243.7,3427927.95 5793243.75,3427927.95 5793243.8,3427<br>927.95 5793243.85,3427928 5793243.85,3427928 5793243.8,3427928.05 5793243.8,<br>3427928.05 5793243.75)),((3427927.95 5793243.75,3427927.95 5793243.7,3427927.9 ←<br>5793243.7,3427927.85 5793243.7,<br>3427927.85 5793243.75,3427927.85 5793243.8,3427927.85 5793243.85,3427927.9 5793243.85,<br>3427927.95 5793243.85,3427927.95 5793243.8,3427927.95 5793243.75))) |

```

--- Checking all the pixels of a large raster tile can take a long time.
--- You can dramatically improve speed at some lose of precision by orders of magnitude
-- by sampling pixels using the step optional parameter of generate_series.
-- This next example does the same as previous but by checking 1 for every 4 (2x2) pixels ←
-- and putting in the last checked
-- putting in the checked pixel as the value for subsequent 4

SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
 ST_UpperLeftX(rast), ST_UpperLeftY(rast),
 ST_UpperLeftX(rast) + ST_ScaleX(rast)*2,
 ST_UpperLeftY(rast) + ST_ScaleY(rast)*2, 0
), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
FROM dummy_rast CROSS JOIN
generate_series(1,1000,2) As x CROSS JOIN generate_series(1,1000,2) As y
WHERE rid = 2
 AND x <= ST_Width(rast) AND y <= ST_Height(rast)) As foo
WHERE
 ST_Intersects(
 pixpolyg,
 ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928 ←
 5793243.75,3427928 5793244))',0)
) AND b2val != 254;

 shadow

MULTIPOLYGON(((3427927.9 5793243.85,3427927.8 5793243.85,3427927.8 5793243.95,
3427927.9 5793243.95,3427928 5793243.95,3427928.1 5793243.95,3427928.1 5793243.85,3427928 ←
5793243.85,3427927.9 5793243.85)),
((3427927.9 5793243.65,3427927.8 5793243.65,3427927.8 5793243.75,3427927.8 ←
5793243.85,3427927.9 5793243.85,
3427928 5793243.85,3427928 5793243.75,3427928.1 5793243.75,3427928.1 5793243.65,3427928 ←
5793243.65,3427927.9 5793243.65)))

```

### See Also

[ST\\_SetValue](#), [ST\\_DumpAsPolygons](#), [ST\\_NumBands](#), [ST\\_PixelAsPolygon](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_UpperLeftX](#), [ST\\_UpperLeftY](#), [ST\\_SRID](#), [ST\\_AsText](#), [ST\\_Point](#), [ST\\_MakeEnvelope](#), [ST\\_Intersects](#), [ST\\_Intersection](#)

## 10.6.8 ST\_NearestValue

**ST\_NearestValue** — Returns the nearest non-NODATA value of a given band's pixel specified by a columnx and rowy or a geometric point expressed in the same spatial reference coordinate system as the raster.

### Synopsis

```

double precision ST_NearestValue(raster rast, integer bandnum, geometry pt, boolean exclude_nodata_value=true);
double precision ST_NearestValue(raster rast, geometry pt, boolean exclude_nodata_value=true);
double precision ST_NearestValue(raster rast, integer bandnum, integer columnx, integer rowy, boolean exclude_nodata_value=true);
double precision ST_NearestValue(raster rast, integer columnx, integer rowy, boolean exclude_nodata_value=true);

```

### Description

Returns the nearest non-NODATA value of a given band in a given columnx, rowy pixel or at a specific geometric point. If the columnx, rowy pixel or the pixel at the specified geometric point is NODATA, the function will find the nearest pixel to the columnx, rowy pixel or geometric point whose value is not NODATA.

Band numbers start at 1 and bandnum is assumed to be 1 if not specified. If `exclude_nodata_value` is set to false, then all pixels include `nodata` pixels are considered to intersect and return value. If `exclude_nodata_value` is not passed in then reads it from metadata of raster.

Availability: 2.1.0



**Note**  
`ST_NearestValue` is a drop-in replacement for `ST_Value`.

**Examples**

```
-- pixel 2x2 has value
SELECT
 ST_Value(rast, 2, 2) AS value,
 ST_NearestValue(rast, 2, 2) AS nearestvalue
FROM (
 SELECT
 ST_SetValue(
 ST_SetValue(
 ST_SetValue(
 ST_SetValue(
 ST_SetValue(
 ST_AddBand(
 ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
 '8BUI'::text, 1, 0
),
 1, 1, 0.
),
 2, 3, 0.
),
 3, 5, 0.
),
 4, 2, 0.
),
 5, 4, 0.
) AS rast
) AS foo

value | nearestvalue
-----+-----
1 | 1
```

```
-- pixel 2x3 is NODATA
SELECT
 ST_Value(rast, 2, 3) AS value,
 ST_NearestValue(rast, 2, 3) AS nearestvalue
FROM (
 SELECT
 ST_SetValue(
 ST_SetValue(
 ST_SetValue(
 ST_SetValue(
 ST_SetValue(
 ST_AddBand(
 ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
 '8BUI'::text, 1, 0
),
 1, 1, 0.
),
 2, 3, 0.
),
 3, 5, 0.
),
 4, 2, 0.
),
 5, 4, 0.
) AS rast
) AS foo
```

```

 1, 1, 0.
),
 2, 3, 0.
),
 3, 5, 0.
),
 4, 2, 0.
),
 5, 4, 0.
) AS rast
) AS foo

value | nearestvalue
-----+-----
 | 1
```

See Also

[ST\\_Neighborhood](#), [ST\\_Value](#)

10.6.9 ST\_SetZ

ST\_SetZ — Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimension using the requested resample algorithm.

Synopsis

geometry **ST\_SetZ**(raster rast, geometry geom, text resample=nearest, integer band=1);

Description

Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimensions using the requested resample algorithm.

The `resample` parameter can be set to "nearest" to copy the values from the cell each vertex falls within, or "bilinear" to use [bilinear interpolation](#) to calculate a value that takes neighboring cells into account also.

Availability: 3.2.0

Examples

```
--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
WITH test_raster AS (
SELECT
ST_SetValues(
 ST_AddBand(
 ST_MakeEmptyRaster(width =
> 2, height =
> 2,
 upperleftx =
> 0, upperlefty =
```

```

> 2,
 scalex =
> 1.0, scaley =
> -1.0,
 skewx =
> 0, skewy =
> 0, srid =
> 4326),
 index =
> 1, pixeltype =
> '16BSI',
 initialvalue =
> 0,
 nodataval =
> -999),
 1,1,1,
 newvalueset =
> ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8]]) AS rast
)
SELECT
ST_AsText(
 ST_SetZ(
 rast,
 band =
> 1,
 geom =
> 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)::geometry,
 resample =
> 'bilinear'
))
FROM test_raster

 st_astext

LINESTRING Z (1 1.9 38,1 0.2 27)

```

## See Also

[ST\\_Value](#), [ST\\_SetM](#)

### 10.6.10 ST\_SetM

**ST\_SetM** — Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the M dimension using the requested resample algorithm.

#### Synopsis

geometry **ST\_SetM**(raster rast, geometry geom, text resample=nearest, integer band=1);

#### Description

Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the M dimensions using the requested resample algorithm.

The `resample` parameter can be set to "nearest" to copy the values from the cell each vertex falls within, or "bilinear" to use [bilinear interpolation](#) to calculate a value that takes neighboring cells into account also.

Availability: 3.2.0

## Examples

```
--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
WITH test_raster AS (
SELECT
ST_SetValues(
 ST_AddBand(
 ST_MakeEmptyRaster(width =
> 2, height =
> 2,
 upperleftx =
> 0, upperlefty =
> 2,
 scalex =
> 1.0, scaley =
> -1.0,
 skewx =
> 0, skewy =
> 0, srid =
> 4326),
 index =
> 1, pixeltype =
> '16BSI',
 initialvalue =
> 0,
 nodataval =
> -999),
 1,1,1,
 newvalueset =
>ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8]]) AS rast
)
SELECT
ST_AsText(
 ST_SetM(
 rast,
 band =
> 1,
 geom =
> 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)::geometry,
 resample =
> 'bilinear'
))
FROM test_raster

 st_astext

LINESTRING M (1 1.9 38,1 0.2 27)
```

## See Also

[ST\\_Value](#), [ST\\_SetZ](#)

## 10.6.11 ST\_Neighborhood

**ST\_Neighborhood** — Returns a 2-D double precision array of the non-NODATA values around a given band's pixel specified by either a columnX and rowY or a geometric point expressed in the same spatial reference coordinate system as the raster.

### Synopsis

```
double precision[][] ST_Neighborhood(raster rast, integer bandnum, integer columnX, integer rowY, integer distanceX, integer
distanceY, boolean exclude_nodata_value=true);
double precision[][] ST_Neighborhood(raster rast, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean
exclude_nodata_value=true);
double precision[][] ST_Neighborhood(raster rast, integer bandnum, geometry pt, integer distanceX, integer distanceY, boolean
exclude_nodata_value=true);
double precision[][] ST_Neighborhood(raster rast, geometry pt, integer distanceX, integer distanceY, boolean exclude_nodata_value=true);
```

### Description

Returns a 2-D double precision array of the non-NODATA values around a given band's pixel specified by either a columnX and rowY or a geometric point expressed in the same spatial reference coordinate system as the raster. The `distanceX` and `distanceY` parameters define the number of pixels around the specified pixel in the X and Y axes, e.g. I want all values within 3 pixel distance along the X axis and 2 pixel distance along the Y axis around my pixel of interest. The center value of the 2-D array will be the value at the pixel specified by the columnX and rowY or the geometric point.

Band numbers start at 1 and `bandnum` is assumed to be 1 if not specified. If `exclude_nodata_value` is set to false, then all pixels include nodata pixels are considered to intersect and return value. If `exclude_nodata_value` is not passed in then reads it from metadata of raster.



#### Note

The number of elements along each axis of the returning 2-D array is  $2 * (\text{distanceX}|\text{distanceY}) + 1$ . So for a `distanceX` and `distanceY` of 1, the returning array will be 3x3.



#### Note

The 2-D array output can be passed to any of the raster processing builtin functions, e.g. `ST_Min4ma`, `ST_Sum4ma`, `ST_Mean4ma`.

Availability: 2.1.0

### Examples

```
-- pixel 2x2 has value
SELECT
 ST_Neighborhood(rast, 2, 2, 1, 1)
FROM (
 SELECT
 ST_SetValues(
 ST_AddBand(
 ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
 '8BUI'::text, 1, 0
),
 1, 1, 1, ARRAY[
 [0, 1, 1, 1, 1],
 [1, 1, 1, 0, 1],
```



```

 [1, 0, 1, 1, 1],
 [1, 1, 1, 1, 0],
 [1, 1, 0, 1, 1]
]::double precision[],
 1
) AS rast
) AS foo

 st_neighborhood

{{NULL,1,1},{1,1,1},{1,NULL,1}}

```

```

-- pixel 2x3 is NODATA
SELECT
 ST_Neighborhood(rast, 2, 3, 1, 1)
FROM (
 SELECT
 ST_SetValues(
 ST_AddBand(
 ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
 '8BUI'::text, 1, 0
),
 1, 1, 1, ARRAY[
 [0, 1, 1, 1, 1],
 [1, 1, 1, 0, 1],
 [1, 0, 1, 1, 1],
 [1, 1, 1, 1, 0],
 [1, 1, 0, 1, 1]
]::double precision[],
 1
) AS rast
) AS foo

 st_neighborhood

{{1,1,1},{1,NULL,1},{1,1,1}}

```

```

-- pixel 3x3 has value
-- exclude_nodata_value = FALSE
SELECT
 ST_Neighborhood(rast, 3, 3, 1, 1, false)
FROM ST_SetValues(
 ST_AddBand(
 ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
 '8BUI'::text, 1, 0
),
 1, 1, 1, ARRAY[
 [0, 1, 1, 1, 1],
 [1, 1, 1, 0, 1],
 [1, 0, 1, 1, 1],
 [1, 1, 1, 1, 0],
 [1, 1, 0, 1, 1]
]::double precision[],
 1
) AS rast

 st_neighborhood

{{1,1,0},{0,1,1},{1,1,1}}

```

**See Also**

[ST\\_NearestValue](#), [ST\\_Min4ma](#), [ST\\_Max4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Range4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

**10.6.12 ST\_SetValue**

**ST\_SetValue** — Returns modified raster resulting from setting the value of a given band in a given columnx, rowy pixel or the pixels that intersect a particular geometry. Band numbers start at 1 and assumed to be 1 if not specified.

**Synopsis**

```
raster ST_SetValue(raster rast, integer bandnum, geometry geom, double precision newvalue);
raster ST_SetValue(raster rast, geometry geom, double precision newvalue);
raster ST_SetValue(raster rast, integer bandnum, integer columnx, integer rowy, double precision newvalue);
raster ST_SetValue(raster rast, integer columnx, integer rowy, double precision newvalue);
```

**Description**

Returns modified raster resulting from setting the specified pixels' values to new value for the designated band given the raster's row and column or a geometry. If no band is specified, then band 1 is assumed.

Enhanced: 2.1.0 Geometry variant of **ST\_SetValue**() now supports any geometry type, not just point. The geometry variant is a wrapper around the **geomval[]** variant of **ST\_SetValues**()

**Examples**

```
-- Geometry example
SELECT (foo.geomval).val, ST_AsText(ST_Union((foo.geomval).geom))
FROM (SELECT ST_DumpAsPolygons(
 ST_SetValue(rast,1,
 ST_Point(3427927.75, 5793243.95),
 50)
) As geomval
FROM dummy_rast
where rid = 2) As foo
WHERE (foo.geomval).val < 250
GROUP BY (foo.geomval).val;
```

| val | st_astext                                                              |
|-----|------------------------------------------------------------------------|
| 50  | POLYGON((3427927.75 5793244,3427927.75 5793243.95,3427927.8 579324 ... |
| 249 | POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 57932 ... |

```
-- Store the changed raster --
UPDATE dummy_rast SET rast = ST_SetValue(rast,1, ST_Point(3427927.75, 5793243.95),100)
WHERE rid = 2 ;
```

**See Also**

[ST\\_Value](#), [ST\\_DumpAsPolygons](#)

### 10.6.13 ST\_SetValues

**ST\_SetValues** — Returns modified raster resulting from setting the values of a given band.

#### Synopsis

```
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, boolean[][] noset=NULL, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, double precision nosetvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, geomval[] geomvalset, boolean keepnodata=FALSE);
```

#### Description

Returns modified raster resulting from setting specified pixels to new value(s) for the designated band. `columnx` and `rowy` are 1-indexed.

If `keepnodata` is TRUE, those pixels whose values are NODATA will not be set with the corresponding value in `newvalueset`.

For Variant 1, the specific pixels to be set are determined by the `columnx`, `rowy` pixel coordinates and the dimensions of the `newvalueset` array. `noset` can be used to prevent pixels with values present in `newvalueset` from being set (due to PostgreSQL not permitting ragged/jagged arrays). See example Variant 1.

Variant 2 is like Variant 1 but with a simple double precision `nosetvalue` instead of a boolean `noset` array. Elements in `newvalueset` with the `nosetvalue` value will be skipped. See example Variant 2.

For Variant 3, the specific pixels to be set are determined by the `columnx`, `rowy` pixel coordinates, `width` and `height`. See example Variant 3.

Variant 4 is the same as Variant 3 with the exception that it assumes that the first band's pixels of `rast` will be set.

For Variant 5, an array of **geomval** is used to determine the specific pixels to be set. If all the geometries in the array are of type POINT or MULTIPOINT, the function uses a shortcut where the longitude and latitude of each point is used to set a pixel directly. Otherwise, the geometries are converted to rasters and then iterated through in one pass. See example Variant 5.

Availability: 2.1.0

#### Examples: Variant 1

```
/*
The ST_SetValues() does the following...

+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 1 | 1 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | =
> | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
*/
SELECT
 (poly).x,
 (poly).y,
 (poly).val
FROM (
```

```

SELECT
 ST_PixelAsPolygons(
 ST_SetValues(
 ST_AddBand(
 ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
 1, '8BUI', 1, 0
),
 1, 2, 2, ARRAY[[9, 9], [9, 9]]::double precision[]
)
) AS poly
) foo
ORDER BY 1, 2;

```

| x | y | val |
|---|---|-----|
| 1 | 1 | 1   |
| 1 | 2 | 1   |
| 1 | 3 | 1   |
| 2 | 1 | 1   |
| 2 | 2 | 9   |
| 2 | 3 | 9   |
| 3 | 1 | 1   |
| 3 | 2 | 9   |
| 3 | 3 | 9   |

```

/*
The ST_SetValues() does the following...

```

|               |   |               |
|---------------|---|---------------|
| + - + - + - + |   | + - + - + - + |
| 1   1   1     |   | 9   9   9     |
| + - + - + - + |   | + - + - + - + |
| 1   1   1     | = |               |
| >   9   9     |   |               |
| + - + - + - + |   | + - + - + - + |
| 1   1   1     |   | 9   9   9     |
| + - + - + - + |   | + - + - + - + |

```

*/
SELECT
 (poly).x,
 (poly).y,
 (poly).val
FROM (
 SELECT
 ST_PixelAsPolygons(
 ST_SetValues(
 ST_AddBand(
 ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
 1, '8BUI', 1, 0
),
 1, 1, 1, ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[]
)
) AS poly
) foo
ORDER BY 1, 2;

```

| x | y | val |
|---|---|-----|
| 1 | 1 | 9   |
| 1 | 2 | 9   |
| 1 | 3 | 9   |
| 2 | 1 | 9   |
| 2 | 2 |     |

```

2 | 3 | 9
3 | 1 | 9
3 | 2 | 9
3 | 3 | 9

```

```

/*
The ST_SetValues() does the following...

+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 9 | 9 | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | =
> | 1 | | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 9 | 9 | 9 |
+ - + - + - + + - + - + - +
*/
SELECT
 (poly).x,
 (poly).y,
 (poly).val
FROM (
SELECT
 ST_PixelAsPolygons(
 ST_SetValues(
 ST_AddBand(
 ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
 1, '8BUI', 1, 0
),
 1, 1, 1,
 ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[],
 ARRAY[[false], [true]]::boolean[]
)
) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+---
 1 | 1 | 9
 1 | 2 | 1
 1 | 3 | 9
 2 | 1 | 9
 2 | 2 |
 2 | 3 | 9
 3 | 1 | 9
 3 | 2 | 9
 3 | 3 | 9

```

```

/*
The ST_SetValues() does the following...

+ - + - + - + + - + - + - +
| | 1 | 1 | | | 9 | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | =
> | 1 | | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 9 | 9 | 9 |
+ - + - + - + + - + - + - +
*/

```

```

SELECT
 (poly).x,
 (poly).y,
 (poly).val
FROM (
SELECT
 ST_PixelAsPolygons(
 ST_SetValues(
 ST_SetValue(
 ST_AddBand(
 ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
 1, '8BUI', 1, 0
),
 1, 1, 1, NULL
),
 1, 1, 1,
 ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
 ARRAY[[false], [true]]::boolean[][],
 TRUE
)
) AS poly
) foo
ORDER BY 1, 2;

```

```

x | y | val
---+---+-----
1 | 1 |
1 | 2 | 1
1 | 3 | 9
2 | 1 | 9
2 | 2 |
2 | 3 | 9
3 | 1 | 9
3 | 2 | 9
3 | 3 | 9

```

### Examples: Variant 2

```

/*
The ST_SetValues() does the following...

```

```

+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 1 | 1 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | =
> | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
*/

```

```

SELECT
 (poly).x,
 (poly).y,
 (poly).val
FROM (
SELECT
 ST_PixelAsPolygons(
 ST_SetValues(
 ST_AddBand(
 ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),

```

```

 1, '8BUI', 1, 0
),
 1, 1, 1, ARRAY[[-1, -1, -1], [-1, 9, 9], [-1, 9, 9]]::double precision[], -1
)
) AS poly
) foo
ORDER BY 1, 2;

```

| x | y | val |
|---|---|-----|
| 1 | 1 | 1   |
| 1 | 2 | 1   |
| 1 | 3 | 1   |
| 2 | 1 | 1   |
| 2 | 2 | 9   |
| 2 | 3 | 9   |
| 3 | 1 | 1   |
| 3 | 2 | 9   |
| 3 | 3 | 9   |

```

/*
This example is like the previous one. Instead of nosetvalue = -1, nosetvalue = NULL

```

The ST\_SetValues() does the following...

|               |   |               |
|---------------|---|---------------|
| + - + - + - + |   | + - + - + - + |
| 1   1   1     |   | 1   1   1     |
| + - + - + - + |   | + - + - + - + |
| 1   1   1     | = |               |
| >   1   9   9 |   |               |
| + - + - + - + |   | + - + - + - + |
| 1   1   1     |   | 1   9   9     |
| + - + - + - + |   | + - + - + - + |

```

*/
SELECT
 (poly).x,
 (poly).y,
 (poly).val
FROM (
 SELECT
 ST_PixelAsPolygons(
 ST_SetValues(
 ST_AddBand(
 ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
 1, '8BUI', 1, 0
),
 1, 1, 1, ARRAY[NULL, NULL, NULL], [NULL, 9, 9], [NULL, 9, 9]]::double ←
 precision[], NULL::double precision
)
) AS poly
) foo
ORDER BY 1, 2;

```

| x | y | val |
|---|---|-----|
| 1 | 1 | 1   |
| 1 | 2 | 1   |
| 1 | 3 | 1   |
| 2 | 1 | 1   |
| 2 | 2 | 9   |
| 2 | 3 | 9   |
| 3 | 1 | 1   |

```

3 | 2 | 9
3 | 3 | 9

```

### Examples: Variant 3

```

/*
The ST_SetValues() does the following...

+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 1 | 1 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | =
> | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
*/
SELECT
 (poly).x,
 (poly).y,
 (poly).val
FROM (
 SELECT
 ST_PixelAsPolygons(
 ST_SetValues(
 ST_AddBand(
 ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
 1, '8BUI', 1, 0
),
 1, 2, 2, 2, 2, 9
)
) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+---
 1 | 1 | 1
 1 | 2 | 1
 1 | 3 | 1
 2 | 1 | 1
 2 | 2 | 9
 2 | 3 | 9
 3 | 1 | 1
 3 | 2 | 9
 3 | 3 | 9

```

```

/*
The ST_SetValues() does the following...

+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 1 | 1 |
+ - + - + - + + - + - + - +
| 1 | | 1 | =
> | 1 | | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
*/
SELECT

```



```

 (poly).x,
 (poly).y,
 (poly).val
FROM (
SELECT
 ST_PixelAsPolygons(
 ST_SetValues(
 ST_SetValue(
 ST_AddBand(
 ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
 1, '8BUI', 1, 0
),
 1, 2, 2, NULL
),
 1, 2, 2, 2, 2, 9, TRUE
)
) AS poly
) foo
ORDER BY 1, 2;
```

| x | y | val |
|---|---|-----|
| 1 | 1 | 1   |
| 1 | 2 | 1   |
| 1 | 3 | 1   |
| 2 | 1 | 1   |
| 2 | 2 |     |
| 2 | 3 | 9   |
| 3 | 1 | 1   |
| 3 | 2 | 9   |
| 3 | 3 | 9   |

Examples: Variant 5

```

WITH foo AS (
 SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', 0, 0) AS rast
), bar AS (
 SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)::geometry geom UNION ALL
 SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))::geometry geom UNION ALL
 SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))::geometry geom UNION ALL
 SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)::geometry
)
SELECT
 rid, gid, ST_DumpValues(ST_SetValue(rast, 1, geom, gid))
FROM foo t1
CROSS JOIN bar t2
ORDER BY rid, gid;
```

| rid | gid | st_dumpvalues                                                                                                                                       |
|-----|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| 1   | 1   | (1, "{ {NULL,NULL,NULL,NULL,NULL}, {NULL,NULL,NULL,NULL,NULL}, {NULL,NULL,1,NULL, NULL}, {NULL,NULL,NULL,NULL,NULL}, {NULL,NULL,NULL,NULL,NULL} }") |
| 1   | 2   | (1, "{ {NULL,NULL,NULL,NULL,NULL}, {NULL,2,2,2,NULL}, {NULL,2,2,2,NULL}, {2,2,2,2,NULL}, {NULL,NULL,NULL,NULL,NULL} }")                             |
| 1   | 3   | (1, "{ {3,3,3,3,3}, {3,NULL,NULL,NULL,NULL}, {3,NULL,NULL,NULL,NULL}, {3,NULL,NULL, NULL,NULL}, {NULL,NULL,NULL,NULL,NULL} }")                      |

```
1 | 4 | (1, "{4,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL, ←
NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,4}}")
(4 rows)
```

The following shows that geomvals later in the array can overwrite prior geomvals

```
WITH foo AS (
 SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
 0, 0) AS rast
), bar AS (
 SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)::geometry geom UNION ALL
 SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))::geometry geom UNION ←
 ALL
 SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))::geometry ←
 geom UNION ALL
 SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)::geometry
)
SELECT
 t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2.gid), ←
 ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 1
 AND t3.gid = 2
ORDER BY t1.rid, t2.gid, t3.gid;

rid | gid | gid | st_dumpvalues
-----+-----+-----+-----
1 | 1 | 2 | (1, "{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{ ←
NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")
(1 row)
```

This example is the opposite of the prior example

```
WITH foo AS (
 SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
 0, 0) AS rast
), bar AS (
 SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)::geometry geom UNION ALL
 SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))::geometry geom UNION ←
 ALL
 SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))::geometry ←
 geom UNION ALL
 SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)::geometry
)
SELECT
 t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2.gid), ←
 ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 2
 AND t3.gid = 1
ORDER BY t1.rid, t2.gid, t3.gid;

rid | gid | gid | st_dumpvalues
-----+-----+-----+-----
1 | 2 | 1 | (1, "{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,1,2,NULL},{ ←
NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")
```

(1 row)

See Also

[ST\\_Value](#), [ST\\_SetValue](#), [ST\\_PixelAsPolygons](#)

10.6.14 ST\_DumpValues

ST\_DumpValues — Get the values of the specified band as a 2-dimension array.

Synopsis

setof record **ST\_DumpValues**( raster rast , integer[] nband=NULL , boolean exclude\_nodata\_value=true );  
double precision[][] **ST\_DumpValues**( raster rast , integer nband , boolean exclude\_nodata\_value=true );

Description

Get the values of the specified band as a 2-dimension array (first index is row, second is column). If nband is NULL or not provided, all raster bands are processed.

Availability: 2.1.0

Examples

```
WITH foo AS (
 SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), ←
 1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
 (ST_DumpValues(rast)).*
FROM foo;
```

| nband | valarray                                             |
|-------|------------------------------------------------------|
| 1     | {{1,1,1},{1,1,1},{1,1,1}}                            |
| 2     | {{3,3,3},{3,3,3},{3,3,3}}                            |
| 3     | {{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}} |

(3 rows)

```
WITH foo AS (
 SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), ←
 1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
 (ST_DumpValues(rast, ARRAY[3, 1])).*
FROM foo;
```

| nband | valarray                                             |
|-------|------------------------------------------------------|
| 3     | {{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}} |
| 1     | {{1,1,1},{1,1,1},{1,1,1}}                            |

(2 rows)

```

WITH foo AS (
 SELECT ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI ←
 ', 1, 0), 1, 2, 5) AS rast
)
SELECT
 (ST_DumpValues(rast, 1))[2][1]
FROM foo;

 st_dumpvalues

 5
(1 row)

```

### See Also

[ST\\_Value](#), [ST\\_SetValue](#), [ST\\_SetValues](#)

## 10.6.15 ST\_PixelOfValue

**ST\_PixelOfValue** — Get the columnx, rowy coordinates of the pixel whose value equals the search value.

### Synopsis

```

setof record ST_PixelOfValue(raster rast , integer nband , double precision[] search , boolean exclude_nodata_value=true);
setof record ST_PixelOfValue(raster rast , double precision[] search , boolean exclude_nodata_value=true);
setof record ST_PixelOfValue(raster rast , integer nband , double precision search , boolean exclude_nodata_value=true);
setof record ST_PixelOfValue(raster rast , double precision search , boolean exclude_nodata_value=true);

```

### Description

Get the columnx, rowy coordinates of the pixel whose value equals the search value. If no band is specified, then band 1 is assumed.

Availability: 2.1.0

### Examples

```

SELECT
 (pixels).*
FROM (
 SELECT
 ST_PixelOfValue(
 ST_SetValue(
 ST_SetValue(
 ST_SetValue(
 ST_SetValue(
 ST_SetValue(
 ST_AddBand(
 ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
 '8BUI'::text, 1, 0
),
 1, 1, 0
),
 2, 3, 0
),
 1, 1, 0
),
 1, 1, 0
),
 1, 1, 0
),
 1, 1, 0
)
)

```

```

 3, 5, 0
),
 4, 2, 0
),
 5, 4, 255
)
, 1, ARRAY[1, 255]) AS pixels
) AS foo

val | x | y
----+---+---
 1 | 1 | 2
 1 | 1 | 3
 1 | 1 | 4
 1 | 1 | 5
 1 | 2 | 1
 1 | 2 | 2
 1 | 2 | 4
 1 | 2 | 5
 1 | 3 | 1
 1 | 3 | 2
 1 | 3 | 3
 1 | 3 | 4
 1 | 4 | 1
 1 | 4 | 3
 1 | 4 | 4
 1 | 4 | 5
 1 | 5 | 1
 1 | 5 | 2
 1 | 5 | 3
255 | 5 | 4
 1 | 5 | 5
```

# 10.7 Raster Editors

## 10.7.1 ST\_SetGeoReference

**ST\_SetGeoReference** — Set Georeference 6 georeference parameters in a single call. Numbers should be separated by white space. Accepts inputs in GDAL or ESRI format. Default is GDAL.

### Synopsis

raster **ST\_SetGeoReference**(raster rast, text georefcoords, text format=GDAL);  
raster **ST\_SetGeoReference**(raster rast, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy);

### Description

Set Georeference 6 georeference parameters in a single call. Accepts inputs in 'GDAL' or 'ESRI' format. Default is GDAL. If 6 coordinates are not provided will return null.

Difference between format representations is as follows:

GDAL:

```
scalex skewy skewx scaley upperleftx upperlefty
```

ESRI:

```
scalex skewx skewy scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5
```



**Note**  
If the raster has out-db bands, changing the georeference may result in incorrect access of the band's externally stored data.

Enhanced: 2.1.0 Addition of ST\_SetGeoReference(raster, double precision, ...) variant

Examples

```
WITH foo AS (
 SELECT ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0) AS rast
)
SELECT
 0 AS rid, (ST_Metadatas(rast)).*
FROM foo
UNION ALL
SELECT
 1, (ST_Metadatas(ST_SetGeoReference(rast, '10 0 0 -10 0.1 0.1', 'GDAL'))).*
FROM foo
UNION ALL
SELECT
 2, (ST_Metadatas(ST_SetGeoReference(rast, '10 0 0 -10 5.1 -4.9', 'ESRI'))).*
FROM foo
UNION ALL
SELECT
 3, (ST_Metadatas(ST_SetGeoReference(rast, 1, 1, 10, -10, 0.001, 0.001))).*
FROM foo
```

| rid   | upperleftx          | upperlefty          | width | height | scalex | scaley | skewx |   |
|-------|---------------------|---------------------|-------|--------|--------|--------|-------|---|
| skewy | srid                | numbands            |       |        |        |        |       | ↔ |
| 0     | 0                   | 0                   | 5     | 5      | 1      | -1     | 0     | ↔ |
| 1     | 0.1                 | 0.1                 | 5     | 5      | 10     | -10    | 0     | ↔ |
| 2     | 0.09999999999999996 | 0.09999999999999996 | 5     | 5      | 10     | -10    | 0     | ↔ |
| 3     | 1                   | 1                   | 5     | 5      | 10     | -10    | 0.001 | ↔ |

See Also

[ST\\_GeoReference](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_UpperLeftX](#), [ST\\_UpperLeftY](#)

10.7.2 ST\_SetRotation

ST\_SetRotation — Set the rotation of the raster in radian.

Synopsis

raster **ST\_SetRotation**(raster rast, float8 rotation);

Description

Uniformly rotate the raster. Rotation is in radian. Refer to [World File](#) for more details.

Examples

```
SELECT
 ST_ScaleX(rast1), ST_ScaleY(rast1), ST_SkewX(rast1), ST_SkewY(rast1),
 ST_ScaleX(rast2), ST_ScaleY(rast2), ST_SkewX(rast2), ST_SkewY(rast2)
FROM (
 SELECT ST_SetRotation(rast, 15) AS rast1, rast as rast2 FROM dummy_rast
) AS foo;
```

| st_scalex           | st_scaley           | st_skewx           | st_skewy           |   |
|---------------------|---------------------|--------------------|--------------------|---|
| st_scalex           | st_scaley           | st_skewx           | st_skewy           |   |
| -1.51937582571764   | -2.27906373857646   | 1.95086352047135   | 1.30057568031423   | ↔ |
| 2                   | 3                   | 0                  | 0                  |   |
| -0.0379843956429411 | -0.0379843956429411 | 0.0325143920078558 | 0.0325143920078558 | ↔ |
| 0.05                | -0.05               | 0                  | 0                  |   |

See Also

[ST\\_Rotation](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_SkewX](#), [ST\\_SkewY](#)

10.7.3 ST\_SetScale

ST\_SetScale — Sets the X and Y size of pixels in units of coordinate reference system. Number units/pixel width/height.

Synopsis

raster **ST\_SetScale**(raster rast, float8 xy);  
raster **ST\_SetScale**(raster rast, float8 x, float8 y);

Description

Sets the X and Y size of pixels in units of coordinate reference system. Number units/pixel width/height. If only one unit passed in, assumed X and Y are the same number.

**Note**

ST\_SetScale is different from [ST\\_Rescale](#) in that ST\_SetScale do not resample the raster to match the raster extent. It only changes the metadata (or georeference) of the raster to correct an originally mis-specified scaling. ST\_Rescale results in a raster having different width and height computed to fit the geographic extent of the input raster. ST\_SetScale do not modify the width, nor the height of the raster.

Changed: 2.0.0 In WKTRaster versions this was called ST\_SetPixelSize. This was changed in 2.0.0.

Examples

```
UPDATE dummy_rast
 SET rast = ST_SetScale(rast, 1.5)
WHERE rid = 2;

SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
WHERE rid = 2;
```

| pixx | pixy | newbox                                            |
|------|------|---------------------------------------------------|
| 1.5  | 1.5  | BOX(3427927.75 5793244 0, 3427935.25 5793251.5 0) |

```
UPDATE dummy_rast
 SET rast = ST_SetScale(rast, 1.5, 0.55)
WHERE rid = 2;

SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
WHERE rid = 2;
```

| pixx | pixy | newbox                                         |
|------|------|------------------------------------------------|
| 1.5  | 0.55 | BOX(3427927.75 5793244 0,3427935.25 5793247 0) |

See Also

[ST\\_ScaleX](#), [ST\\_ScaleY](#), [Box3D](#)

10.7.4 ST\_SetSkew

ST\_SetSkew — Sets the georeference X and Y skew (or rotation parameter). If only one is passed in, sets X and Y to the same value.

Synopsis

raster **ST\_SetSkew**(raster rast, float8 skewxy);  
raster **ST\_SetSkew**(raster rast, float8 skewx, float8 skewy);

Description

Sets the georeference X and Y skew (or rotation parameter). If only one is passed in, sets X and Y to the same value. Refer to [World File](#) for more details.

Examples

```
-- Example 1
UPDATE dummy_rast SET rast = ST_SetSkew(rast,1,2) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
 ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

| rid | skewx | skewy | georef                                       |
|-----|-------|-------|----------------------------------------------|
| 1   | 1     | 2     | SRID=31466; AUTHORITY=EPSG:31466; PROJCS=... |



```
1 | 1 | 2 | 2.0000000000
 : 2.0000000000
 : 1.0000000000
 : 3.0000000000
 : 0.5000000000
 : 0.5000000000

-- Example 2 set both to same number:
UPDATE dummy_rast SET rast = ST_SetSkew(rast,0) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
 ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

| rid | skewx | skewy | georef         |
|-----|-------|-------|----------------|
| 1   | 0     | 0     | 2.0000000000   |
|     |       |       | : 0.0000000000 |
|     |       |       | : 0.0000000000 |
|     |       |       | : 3.0000000000 |
|     |       |       | : 0.5000000000 |
|     |       |       | : 0.5000000000 |

**See Also**

[ST\\_GeoReference](#), [ST\\_SetGeoReference](#), [ST\\_SkewX](#), [ST\\_SkewY](#)

### 10.7.5 ST\_SetSRID

**ST\_SetSRID** — Sets the SRID of a raster to a particular integer srid defined in the spatial\_ref\_sys table.

**Synopsis**

raster **ST\_SetSRID**(raster rast, integer srid);

**Description**

Sets the SRID on a raster to a particular integer value.



**Note**  
This function does not transform the raster in any way - it simply sets meta data defining the spatial ref of the coordinate reference system that it's currently in. Useful for transformations later.

**See Also**

Section [4.5](#), [ST\\_SRID](#)

### 10.7.6 ST\_SetUpperLeft

**ST\_SetUpperLeft** — Sets the value of the upper left corner of the pixel of the raster to projected X and Y coordinates.

## Synopsis

raster **ST\_SetUpperLeft**(raster rast, double precision x, double precision y);

## Description

Set the value of the upper left corner of raster to the projected X and Y coordinates

## Examples

```
SELECT ST_SetUpperLeft (rast, -71.01, 42.37)
FROM dummy_rast
WHERE rid = 2;
```

## See Also

[ST\\_UpperLeftX](#), [ST\\_UpperLeftY](#)

## 10.7.7 ST\_Resample

**ST\_Resample** — Resample a raster using a specified resampling algorithm, new dimensions, an arbitrary grid corner and a set of raster georeferencing attributes defined or borrowed from another raster.

## Synopsis

raster **ST\_Resample**(raster rast, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor, double precision maxerr=0.125);  
raster **ST\_Resample**(raster rast, double precision scalex=0, double precision scaley=0, double precision gridx=NULL, double precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor, double precision maxerr=0.125);  
raster **ST\_Resample**(raster rast, raster ref, text algorithm=NearestNeighbor, double precision maxerr=0.125, boolean usescale=true);  
raster **ST\_Resample**(raster rast, raster ref, boolean usescale, text algorithm=NearestNeighbor, double precision maxerr=0.125);

## Description

Resample a raster using a specified resampling algorithm, new dimensions (width & height), a grid corner (gridx & gridy) and a set of raster georeferencing attributes (scalex, scaley, skewx & skewy) defined or borrowed from another raster. If using a reference raster, the two rasters must have the same SRID.

New pixel values are computed using one of the following resampling algorithms:

- NearestNeighbor (english or american spelling)
- Bilinear
- Cubic
- CubicSpline
- Lanczos
- Max
- Min

The default is `NearestNeighbor` which is the fastest but results in the worst interpolation.

A `maxerror` percent of 0.125 is used if no `maxerr` is specified.



#### Note

Refer to: [GDAL Warp resampling methods](#) for more details.

Availability: 2.0.0 Requires GDAL 1.6.1+

Enhanced: 3.4.0 max and min resampling options added

### Examples

```
SELECT
 ST_Width(orig) AS orig_width,
 ST_Width(reduce_100) AS new_width
FROM (
 SELECT
 rast AS orig,
 ST_Resample(rast,100,100) AS reduce_100
 FROM aerials.boston
 WHERE ST_Intersects(rast,
 ST_Transform(
 ST_MakeEnvelope(-71.128, 42.2392,-71.1277, 42.2397, 4326),26986)
)
 LIMIT 1
) AS foo;

orig_width | new_width
-----+-----
200 | 100
```

### See Also

[ST\\_Rescale](#), [ST\\_Resize](#), [ST\\_Transform](#)

## 10.7.8 ST\_Rescale

**ST\_Rescale** — Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the `NearestNeighbor` (english or american spelling), `Bilinear`, `Cubic`, `CubicSpline`, `Lanczos`, `Max` or `Min` resampling algorithm. Default is `NearestNeighbor`.

### Synopsis

```
raster ST_Rescale(raster rast, double precision scalexy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_Rescale(raster rast, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

## Description

Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using one of the following resampling algorithms:

- NearestNeighbor (english or american spelling)
- Bilinear
- Cubic
- CubicSpline
- Lanczos
- Max
- Min

The default is NearestNeighbor which is the fastest but results in the worst interpolation.

`scalex` and `scaley` define the new pixel size. `scaley` must often be negative to get well oriented raster.

When the new `scalex` or `scaley` is not a divisor of the raster width or height, the extent of the resulting raster is expanded to encompass the extent of the provided raster. If you want to be sure to retain exact input extent see [ST\\_Resize](#)

`maxerr` is the threshold for transformation approximation by the resampling algorithm (in pixel units). A default of 0.125 is used if no `maxerr` is specified, which is the same value used in GDAL `gdalwarp` utility. If set to zero, no approximation takes place.



### Note

Refer to: [GDAL Warp resampling methods](#) for more details.



### Note

`ST_Rescale` is different from [ST\\_SetScale](#) in that `ST_SetScale` do not resample the raster to match the raster extent. `ST_SetScale` only changes the metadata (or georeference) of the raster to correct an originally mis-specified scaling. `ST_Rescale` results in a raster having different width and height computed to fit the geographic extent of the input raster. `ST_SetScale` do not modify the width, nor the height of the raster.

Availability: 2.0.0 Requires GDAL 1.6.1+

Enhanced: 3.4.0 max and min resampling options added

Changed: 2.1.0 Works on rasters with no SRID

## Examples

A simple example rescaling a raster from a pixel size of 0.001 degree to a pixel size of 0.0015 degree.

```
-- the original raster pixel size
SELECT ST_PixelWidth(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269), '8BUI'::text, 1, 0)) width

width

0.001
```

```
-- the rescaled raster raster pixel size
SELECT ST_PixelWidth(ST_Rescale(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, ↵
 -0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0015)) width

 width

0.0015
```

## See Also

[ST\\_Resize](#), [ST\\_Resample](#), [ST\\_SetScale](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_Transform](#)

## 10.7.9 ST\_Reskew

**ST\_Reskew** — Resample a raster by adjusting only its skew (or rotation parameters). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.

### Synopsis

```
raster ST_Reskew(raster rast, double precision skewxy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_Reskew(raster rast, double precision skewx, double precision skewy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

### Description

Resample a raster by adjusting only its skew (or rotation parameters). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. The default is NearestNeighbor which is the fastest but results in the worst interpolation.

`skewx` and `skewy` define the new skew.

The extent of the new raster will encompass the extent of the provided raster.

A `maxerror` percent of 0.125 if no `maxerr` is specified.



#### Note

Refer to: [GDAL Warp resampling methods](#) for more details.



#### Note

**ST\_Reskew** is different from **ST\_SetSkew** in that **ST\_SetSkew** do not resample the raster to match the raster extent. **ST\_SetSkew** only changes the metadata (or georeference) of the raster to correct an originally mis-specified skew. **ST\_Reskew** results in a raster having different width and height computed to fit the geographic extent of the input raster. **ST\_SetSkew** do not modify the width, nor the height of the raster.

Availability: 2.0.0 Requires GDAL 1.6.1+

Changed: 2.1.0 Works on rasters with no SRID

## Examples

A simple example reskewing a raster from a skew of 0.0 to a skew of 0.0015.

```
-- the original raster non-rotated
SELECT ST_Rotation(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));

-- result
0

-- the reskewed raster raster rotation
SELECT ST_Rotation(ST_Reskew(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, ←
0, 0, 4269), '8BUI'::text, 1, 0), 0.0015));

-- result
-0.982793723247329
```

## See Also

[ST\\_Resample](#), [ST\\_Rescale](#), [ST\\_SetSkew](#), [ST\\_SetRotation](#), [ST\\_SkewX](#), [ST\\_SkewY](#), [ST\\_Transform](#)

## 10.7.10 ST\_SnapToGrid

**ST\_SnapToGrid** — Resample a raster by snapping it to a grid. New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.

## Synopsis

```
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex=DEFAULT 0, double precision scaley=DEFAULT 0);
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, double precision scalexy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

## Description

Resample a raster by snapping it to a grid defined by an arbitrary pixel corner (gridx & gridy) and optionally a pixel size (scalex & scaley). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. The default is NearestNeighbor which is the fastest but results in the worst interpolation.

gridx and gridy define any arbitrary pixel corner of the new grid. This is not necessarily the upper left corner of the new raster and it does not have to be inside or on the edge of the new raster extent.

You can optionally define the pixel size of the new grid with scalex and scaley.

The extent of the new raster will encompass the extent of the provided raster.

A maxerror percent of 0.125 if no maxerr is specified.



### Note

Refer to: [GDAL Warp resampling methods](#) for more details.

**Note**

Use [ST\\_Resample](#) if you need more control over the grid parameters.

Availability: 2.0.0 Requires GDAL 1.6.1+

Changed: 2.1.0 Works on rasters with no SRID

**Examples**

A simple example snapping a raster to a slightly different grid.

```
-- the original raster upper left X
SELECT ST_UpperLeftX(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));
-- result
0

-- the upper left of raster after snapping
SELECT ST_UpperLeftX(ST_SnapToGrid(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, ←
-0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0002, 0.0002));

--result
-0.0008
```

**See Also**

[ST\\_Resample](#), [ST\\_Rescale](#), [ST\\_UpperLeftX](#), [ST\\_UpperLeftY](#)

**10.7.11 ST\_Resize**

**ST\_Resize** — Resize a raster to a new width/height

**Synopsis**

raster **ST\_Resize**(raster rast, integer width, integer height, text algorithm=NearestNeighbor, double precision maxerr=0.125);  
 raster **ST\_Resize**(raster rast, double precision percentwidth, double precision percentheight, text algorithm=NearestNeighbor, double precision maxerr=0.125);  
 raster **ST\_Resize**(raster rast, text width, text height, text algorithm=NearestNeighbor, double precision maxerr=0.125);

**Description**

Resize a raster to a new width/height. The new width/height can be specified in exact number of pixels or a percentage of the raster's width/height. The extent of the the new raster will be the same as the extent of the provided raster.

New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. The default is NearestNeighbor which is the fastest but results in the worst interpolation.

Variant 1 expects the actual width/height of the output raster.

Variant 2 expects decimal values between zero (0) and one (1) indicating the percentage of the input raster's width/height.

Variant 3 takes either the actual width/height of the output raster or a textual percentage ("20%") indicating the percentage of the input raster's width/height.

Availability: 2.1.0 Requires GDAL 1.6.1+

Examples

```
WITH foo AS (
SELECT
 1 AS rid,
 ST_Resize(
 ST_AddBand(
 ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
 , 1, '8BUI', 255, 0
)
 , '50%', '500') AS rast
UNION ALL
SELECT
 2 AS rid,
 ST_Resize(
 ST_AddBand(
 ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
 , 1, '8BUI', 255, 0
)
 , 500, 100) AS rast
UNION ALL
SELECT
 3 AS rid,
 ST_Resize(
 ST_AddBand(
 ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
 , 1, '8BUI', 255, 0
)
 , 0.25, 0.9) AS rast
), bar AS (
 SELECT rid, ST_Metadata(rast) AS meta, rast FROM foo
)
SELECT rid, (meta).* FROM bar
```

| rid      | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | ← |
|----------|------------|------------|-------|--------|--------|--------|-------|-------|------|---|
| numbands |            |            |       |        |        |        |       |       |      |   |
| 1        | 0          | 0          | 500   | 500    | 1      | -1     | 0     | 0     | 0    | ← |
|          | 1          |            |       |        |        |        |       |       |      |   |
| 2        | 0          | 0          | 500   | 100    | 1      | -1     | 0     | 0     | 0    | ← |
|          | 1          |            |       |        |        |        |       |       |      |   |
| 3        | 0          | 0          | 250   | 900    | 1      | -1     | 0     | 0     | 0    | ← |
|          | 1          |            |       |        |        |        |       |       |      |   |
| (3 rows) |            |            |       |        |        |        |       |       |      |   |

See Also

[ST\\_Resample](#), [ST\\_Rescale](#), [ST\\_Reskew](#), [ST\\_SnapToGrid](#)

10.7.12 ST\_Transform

ST\_Transform — Reprojects a raster in a known spatial reference system to another known spatial reference system using specified resampling algorithm. Options are NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos defaulting to NearestNeighbor.

Synopsis

raster **ST\_Transform**(raster rast, integer srid, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex, double precision scaley);



raster **ST\_Transform**(raster rast, integer srid, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);  
 raster **ST\_Transform**(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision maxerr=0.125);

## Description

Reprojects a raster in a known spatial reference system to another known spatial reference system using specified pixel warping algorithm. Uses 'NearestNeighbor' if no algorithm is specified and maxerror percent of 0.125 if no maxerr is specified.

Algorithm options are: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', and 'Lanczos'. Refer to: [GDAL Warp resampling methods](#) for more details.

ST\_Transform is often confused with ST\_SetSRID(). ST\_Transform actually changes the coordinates of a raster (and resamples the pixel values) from one spatial reference system to another, while ST\_SetSRID() simply changes the SRID identifier of the raster.

Unlike the other variants, Variant 3 requires a reference raster as alignto. The transformed raster will be transformed to the spatial reference system (SRID) of the reference raster and be aligned (ST\_SameAlignment = TRUE) to the reference raster.

### Note



If you find your transformation support is not working right, you may need to set the environment variable PROJSO to the .so or .dll projection library your PostGIS is using. This just needs to have the name of the file. So for example on windows, you would in Control Panel -> System -> Environment Variables add a system variable called PROJSO and set it to libproj.dll (if you are using proj 4.6.1). You'll have to restart your PostgreSQL service/daemon after this change.



### Warning

When transforming a coverage of tiles, you almost always want to use a reference raster to insure same alignment and no gaps in your tiles as demonstrated in example: Variant 3.

Availability: 2.0.0 Requires GDAL 1.6.1+

Enhanced: 2.1.0 Addition of ST\_Transform(rast, alignto) variant

## Examples

```
SELECT ST_Width(mass_stm) As w_before, ST_Width(wgs_84) As w_after,
 ST_Height(mass_stm) As h_before, ST_Height(wgs_84) As h_after
FROM
 (SELECT rast As mass_stm, ST_Transform(rast,4326) As wgs_84
 , ST_Transform(rast,4326, 'Bilinear') AS wgs_84_bilin
 FROM aerials.o_2_boston
 WHERE ST_Intersects(rast,
 ST_Transform(ST_MakeEnvelope(-71.128, 42.2392,-71.1277, 42.2397, 4326) ↵
 ,26986))
 LIMIT 1) As foo;
```

| w_before | w_after | h_before | h_after |
|----------|---------|----------|---------|
| 200      | 228     | 200      | 170     |



### Examples: Variant 3

The following shows the difference between using `ST_Transform(raster, srid)` and `ST_Transform(raster, alignto)`

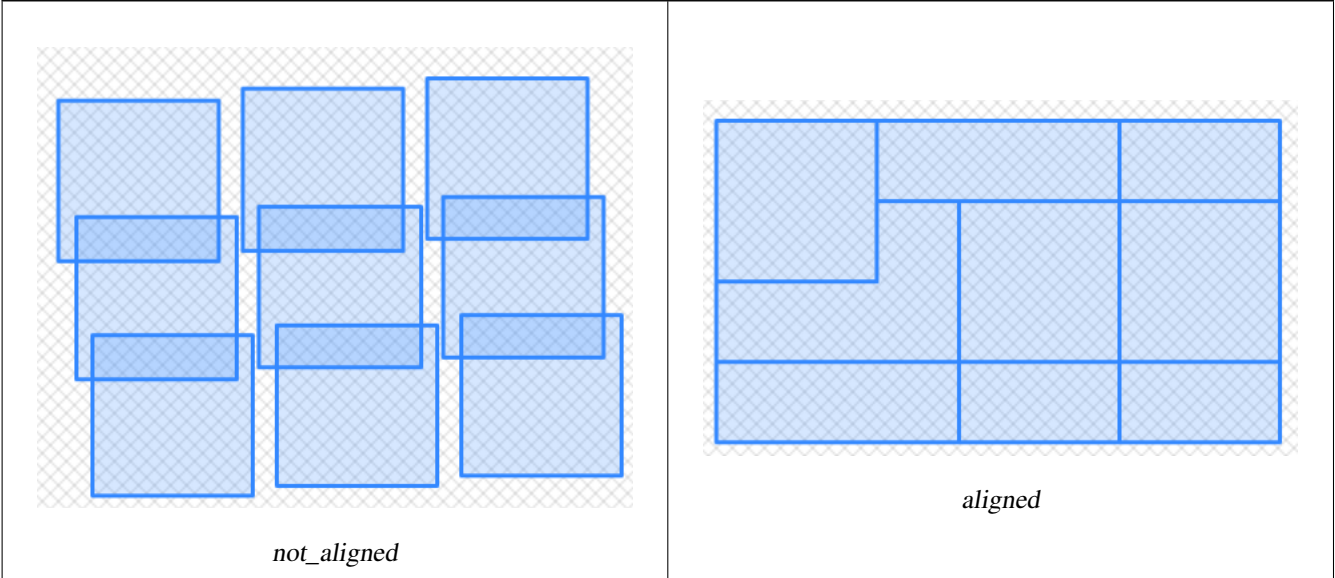
```
WITH foo AS (
 SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 600000, 100, -100, 0, 0, ←
 2163), 1, '16BUI', 1, 0) AS rast UNION ALL
 SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 600000, 100, -100, 0, 0, 2163), ←
 1, '16BUI', 2, 0) AS rast UNION ALL
 SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 600000, 100, -100, 0, 0, 2163), ←
 1, '16BUI', 3, 0) AS rast UNION ALL

 SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599800, 100, -100, 0, 0, 2163), ←
 1, '16BUI', 10, 0) AS rast UNION ALL
 SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599800, 100, -100, 0, 0, 2163), ←
 1, '16BUI', 20, 0) AS rast UNION ALL
 SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599800, 100, -100, 0, 0, 2163), ←
 1, '16BUI', 30, 0) AS rast UNION ALL

 SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599600, 100, -100, 0, 0, 2163), ←
 1, '16BUI', 100, 0) AS rast UNION ALL
 SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599600, 100, -100, 0, 0, 2163), ←
 1, '16BUI', 200, 0) AS rast UNION ALL
 SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599600, 100, -100, 0, 0, 2163), ←
 1, '16BUI', 300, 0) AS rast
), bar AS (
 SELECT
 ST_Transform(rast, 4269) AS alignto
 FROM foo
 LIMIT 1
), baz AS (
 SELECT
 rid,
 rast,
 ST_Transform(rast, 4269) AS not_aligned,
 ST_Transform(rast, alignto) AS aligned
 FROM foo
 CROSS JOIN bar
)
SELECT
```

```
ST_SameAlignment(rast) AS rast,
ST_SameAlignment(not_aligned) AS not_aligned,
ST_SameAlignment(aligned) AS aligned
FROM baz
```

| rast | not_aligned | aligned |
|------|-------------|---------|
| t    | f           | t       |



**See Also**

[ST\\_Transform](#), [ST\\_SetSRID](#)

## 10.8 Raster Band Editors

### 10.8.1 ST\_SetBandNoDataValue

**ST\_SetBandNoDataValue** — Sets the value for the given band that represents no data. Band 1 is assumed if no band is specified. To mark a band as having no nodata value, set the nodata value = NULL.

**Synopsis**

```
raster ST_SetBandNoDataValue(raster rast, double precision nodatavalue);
raster ST_SetBandNoDataValue(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);
```

**Description**

Sets the value that represents no data for the band. Band 1 is assumed if not specified. This will affect results from [ST\\_Polygon](#), [ST\\_DumpAsPolygons](#), and the [ST\\_PixelAs...](#)() functions.

**Examples**

```

-- change just first band no data value
UPDATE dummy_rast
 SET rast = ST_SetBandNoDataValue(rast,1, 254)
WHERE rid = 2;

-- change no data band value of bands 1,2,3
UPDATE dummy_rast
 SET rast =
 ST_SetBandNoDataValue(
 ST_SetBandNoDataValue(
 ST_SetBandNoDataValue(
 rast,1, 254)
 ,2,99),
 3,108)
 WHERE rid = 2;

-- wipe out the nodata value this will ensure all pixels are considered for all processing ←
functions
UPDATE dummy_rast
 SET rast = ST_SetBandNoDataValue(rast,1, NULL)
WHERE rid = 2;

```

### See Also

[ST\\_BandNoDataValue](#), [ST\\_NumBands](#)

## 10.8.2 ST\_SetBandIsNoData

**ST\_SetBandIsNoData** — Sets the isnodata flag of the band to TRUE.

### Synopsis

raster **ST\_SetBandIsNoData**(raster rast, integer band=1);

### Description

Sets the isnodata flag for the band to true. Band 1 is assumed if not specified. This function should be called only when the flag is considered dirty. That is, when the result calling [ST\\_BandIsNoData](#) is different using TRUE as last argument and without using it

Availability: 2.0.0

### Examples

```

-- Create dummy table with one raster column
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ←
 = 3.
-- In the second band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)

```

```

||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'4' -- hasnodatavalue set to true, isnodata value set to false (when it should be true)
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected false
select st_bandisnodata(rast, 1, TRUE) from dummy_rast where rid = 1; -- Expected true

-- The isnodata flag is dirty. We are going to set it to true
update dummy_rast set rast = st_setbandisnodata(rast, 1) where rid = 1;

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true

```

## See Also

[ST\\_BandNoDataValue](#), [ST\\_NumBands](#), [ST\\_SetBandNoDataValue](#), [ST\\_BandIsNoData](#)

## 10.8.3 ST\_SetBandPath

**ST\_SetBandPath** — Update the external path and band number of an out-db band

## Synopsis

raster **ST\_SetBandPath**(raster rast, integer band, text outdbpath, integer outdbindex, boolean force=false);

## Description

Updates an out-db band's external raster file path and external band number.



### Note

If `force` is set to true, no tests are done to ensure compatibility (e.g. alignment, pixel support) between the external raster file and the PostGIS raster. This mode is intended for file system changes where the external raster resides.



### Note

Internally, this method replaces the PostGIS raster's band at index `band` with a new band instead of updating the existing path information.

Availability: 2.5.0

## Examples

```
WITH foo AS (
 SELECT
 ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↵
 loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
 1 AS query,
 *
FROM ST_BandMetadata(
 (SELECT rast FROM foo),
 ARRAY[1,3,2]::int[]
)
UNION ALL
SELECT
 2,
 *
FROM ST_BandMetadata(
 (
 SELECT
 ST_SetBandPath(
 rast,
 2,
 '/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected2.tif ↵
 ',
 1
) AS rast
 FROM foo
),
 ARRAY[1,3,2]::int[]
)
ORDER BY 1, 2;

query | bandnum | pixeltype | nodatavalue | isoutdb | ↵
 path | ↵
outdbbandnum
```

|   |  |   |  |                                           |  |   |  |                                     |
|---|--|---|--|-------------------------------------------|--|---|--|-------------------------------------|
| 1 |  | 1 |  | 8BUI                                      |  | t |  | /home/pele/devel/geo/postgis-git/ ↩ |
|   |  |   |  | raster/test/regress/loader/Projected.tif  |  |   |  | 1                                   |
| 1 |  | 2 |  | 8BUI                                      |  | t |  | /home/pele/devel/geo/postgis-git/ ↩ |
|   |  |   |  | raster/test/regress/loader/Projected.tif  |  |   |  | 2                                   |
| 1 |  | 3 |  | 8BUI                                      |  | t |  | /home/pele/devel/geo/postgis-git/ ↩ |
|   |  |   |  | raster/test/regress/loader/Projected.tif  |  |   |  | 3                                   |
| 2 |  | 1 |  | 8BUI                                      |  | t |  | /home/pele/devel/geo/postgis-git/ ↩ |
|   |  |   |  | raster/test/regress/loader/Projected.tif  |  |   |  | 1                                   |
| 2 |  | 2 |  | 8BUI                                      |  | t |  | /home/pele/devel/geo/postgis-git/ ↩ |
|   |  |   |  | raster/test/regress/loader/Projected2.tif |  |   |  | 1                                   |
| 2 |  | 3 |  | 8BUI                                      |  | t |  | /home/pele/devel/geo/postgis-git/ ↩ |
|   |  |   |  | raster/test/regress/loader/Projected.tif  |  |   |  | 3                                   |

See Also

[ST\\_BandMetaData](#), [ST\\_SetBandIndex](#)

10.8.4 ST\_SetBandIndex

ST\_SetBandIndex — Update the external band number of an out-db band

Synopsis

raster **ST\_SetBandIndex**(raster rast, integer band, integer outdbindex, boolean force=false);

Description

Updates an out-db band’s external band number. This does not touch the external raster file associated with the out-db band



**Note**  
If `force` is set to true, no tests are done to ensure compatibility (e.g. alignment, pixel support) between the external raster file and the PostGIS raster. This mode is intended for where bands are moved around in the external raster file.



**Note**  
Internally, this method replaces the PostGIS raster’s band at index `band` with a new band instead of updating the existing path information.

Availability: 2.5.0

Examples

```
WITH foo AS (
 SELECT
 ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↩
 loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
 1 AS query,
```

```

*
FROM ST_BandMetadata (
 (SELECT rast FROM foo),
 ARRAY[1,3,2]::int[]
)
UNION ALL
SELECT
 2,
 *
FROM ST_BandMetadata (
 (
 SELECT
 ST_SetBandIndex(
 rast,
 2,
 1
) AS rast
 FROM foo
),
 ARRAY[1,3,2]::int[]
)
ORDER BY 1, 2;

query | bandnum | pixeltype | nodatavalue | isoutdb | ↔
 | path | ↔
-----+-----+-----+-----+-----+-----
1 | 1 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↔
 raster/test/regress/loader/Projected.tif | 1
1 | 2 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↔
 raster/test/regress/loader/Projected.tif | 2
1 | 3 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↔
 raster/test/regress/loader/Projected.tif | 3
2 | 1 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↔
 raster/test/regress/loader/Projected.tif | 1
2 | 2 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↔
 raster/test/regress/loader/Projected.tif | 1
2 | 3 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↔
 raster/test/regress/loader/Projected.tif | 3
```

See Also

[ST\\_BandMetaData](#), [ST\\_SetBandPath](#)

# 10.9 Raster Band Statistics and Analytics

## 10.9.1 ST\_Count

**ST\_Count** — Returns the number of pixels in a given band of a raster or raster coverage. If no band is specified defaults to band 1. If `exclude_nodata_value` is set to true, will only count pixels that are not equal to the nodata value.

Synopsis

`bigint ST_Count(raster rast, integer nband=1, boolean exclude_nodata_value=true);`  
`bigint ST_Count(raster rast, boolean exclude_nodata_value);`



Description

Returns the number of pixels in a given band of a raster or raster coverage. If no band is specified `nband` defaults to 1.



**Note**  
If `exclude_nodata_value` is set to true, will only count pixels with value not equal to the `nodata` value of the raster. Set `exclude_nodata_value` to false to get count all pixels

Changed: 3.1.0 - The `ST_Count(rastertable, rastercolumn, ...)` variants removed. Use `ST_CountAgg` instead.

Availability: 2.0.0

Examples

```
--example will count all pixels not 249 and one will count all pixels. --
SELECT rid, ST_Count(ST_SetBandNoDataValue(rast,249)) As exclude_nodata,
 ST_Count(ST_SetBandNoDataValue(rast,249),false) As include_nodata
FROM dummy_rast WHERE rid=2;
```

| rid | exclude_nodata | include_nodata |
|-----|----------------|----------------|
| 2   | 23             | 25             |

See Also

[ST\\_CountAgg](#), [ST\\_SummaryStats](#), [ST\\_SetBandNoDataValue](#)

10.9.2 ST\_CountAgg

`ST_CountAgg` — Aggregate. Returns the number of pixels in a given band of a set of rasters. If no band is specified defaults to band 1. If `exclude_nodata_value` is set to true, will only count pixels that are not equal to the `NODATA` value.

Synopsis

`bigint ST_CountAgg(raster rast, integer nband, boolean exclude_nodata_value, double precision sample_percent);`  
`bigint ST_CountAgg(raster rast, integer nband, boolean exclude_nodata_value);`  
`bigint ST_CountAgg(raster rast, boolean exclude_nodata_value);`

Description

Returns the number of pixels in a given band of a set of rasters. If no band is specified `nband` defaults to 1.

If `exclude_nodata_value` is set to true, will only count pixels with value not equal to the `NODATA` value of the raster. Set `exclude_nodata_value` to false to get count all pixels

By default will sample all pixels. To get faster response, set `sample_percent` to value between zero (0) and one (1)

Availability: 2.2.0

## Examples

```
WITH foo AS (
 SELECT
 rast.rast
 FROM (
 SELECT ST_SetValue(
 ST_SetValue(
 ST_SetValue(
 ST_AddBand(
 ST_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,0)
 , 1, '64BF', 0, 0
)
 , 1, 1, 1, -10
)
 , 1, 5, 4, 0
)
 , 1, 5, 5, 3.14159
) AS rast
) AS rast
 FULL JOIN (
 SELECT generate_series(1, 10) AS id
) AS id
 ON 1 = 1
)
SELECT
 ST_CountAgg(rast, 1, TRUE)
FROM foo;

 st_countagg

 20
(1 row)
```

## See Also

[ST\\_Count](#), [ST\\_SummaryStats](#), [ST\\_SetBandNoDataValue](#)

## 10.9.3 ST\_Histogram

**ST\_Histogram** — Returns a set of record summarizing a raster or raster coverage data distribution separate bin ranges. Number of bins are autocomputed if not specified.

### Synopsis

SETOF record **ST\_Histogram**(raster rast, integer nband=1, boolean exclude\_nodata\_value=true, integer bins=autocomputed, double precision[] width=NULL, boolean right=false);  
 SETOF record **ST\_Histogram**(raster rast, integer nband, integer bins, double precision[] width=NULL, boolean right=false);  
 SETOF record **ST\_Histogram**(raster rast, integer nband, boolean exclude\_nodata\_value, integer bins, boolean right);  
 SETOF record **ST\_Histogram**(raster rast, integer nband, integer bins, boolean right);

### Description

Returns set of records consisting of min, max, count, percent for a given raster band for each bin. If no band is specified nband defaults to 1.



**Note**  
By default only considers pixel values not equal to the `nodata` value . Set `exclude_nodata_value` to `false` to get count all pixels.

**width double precision[]** width: an array indicating the width of each category/bin. If the number of bins is greater than the number of widths, the widths are repeated.

Example: 9 bins, widths are [a, b, c] will have the output be [a, b, c, a, b, c, a, b, c]

**bins integer** Number of breakouts -- this is the number of records you'll get back from the function if specified. If not specified then the number of breakouts is autocomputed.

**right boolean** compute the histogram from the right rather than from the left (default). This changes the criteria for evaluating a value x from [a, b) to (a, b]

Changed: 3.1.0 Removed `ST_Histogram(table_name, column_name)` variant.

Availability: 2.0.0

**Example: Single raster tile - compute histograms for bands 1, 2, 3 and autocompute bins**

```
SELECT band, (stats).*
FROM (SELECT rid, band, ST_Histogram(rast, band) As stats
 FROM dummy_rast CROSS JOIN generate_series(1,3) As band
 WHERE rid=2) As foo;
```

| band | min   | max   | count | percent |
|------|-------|-------|-------|---------|
| 1    | 249   | 250   | 2     | 0.08    |
| 1    | 250   | 251   | 2     | 0.08    |
| 1    | 251   | 252   | 1     | 0.04    |
| 1    | 252   | 253   | 2     | 0.08    |
| 1    | 253   | 254   | 18    | 0.72    |
| 2    | 78    | 113.2 | 11    | 0.44    |
| 2    | 113.2 | 148.4 | 4     | 0.16    |
| 2    | 148.4 | 183.6 | 4     | 0.16    |
| 2    | 183.6 | 218.8 | 1     | 0.04    |
| 2    | 218.8 | 254   | 5     | 0.2     |
| 3    | 62    | 100.4 | 11    | 0.44    |
| 3    | 100.4 | 138.8 | 5     | 0.2     |
| 3    | 138.8 | 177.2 | 4     | 0.16    |
| 3    | 177.2 | 215.6 | 1     | 0.04    |
| 3    | 215.6 | 254   | 4     | 0.16    |

**Example: Just band 2 but for 6 bins**

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6) As stats
 FROM dummy_rast
 WHERE rid=2) As foo;
```

| min        | max        | count | percent |
|------------|------------|-------|---------|
| 78         | 107.333333 | 9     | 0.36    |
| 107.333333 | 136.666667 | 6     | 0.24    |
| 136.666667 | 166        | 0     | 0       |
| 166        | 195.333333 | 4     | 0.16    |

```
195.333333 | 224.666667 | 1 | 0.04
224.666667 | 254 | 5 | 0.2
(6 rows)

-- Same as previous but we explicitly control the pixel value range of each bin.
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6,ARRAY[0.5,1,4,100,5]) As stats
 FROM dummy_rast
 WHERE rid=2) As foo;
```

| min   | max   | count | percent  |
|-------|-------|-------|----------|
| 78    | 78.5  | 1     | 0.08     |
| 78.5  | 79.5  | 1     | 0.04     |
| 79.5  | 83.5  | 0     | 0        |
| 83.5  | 183.5 | 17    | 0.0068   |
| 183.5 | 188.5 | 0     | 0        |
| 188.5 | 254   | 6     | 0.003664 |

(6 rows)

**See Also**

[ST\\_Count](#), [ST\\_SummaryStats](#), [ST\\_SummaryStatsAgg](#)

**10.9.4 ST\_Quantile**

**ST\_Quantile** — Compute quantiles for a raster or raster table coverage in the context of the sample or population. Thus, a value could be examined to be at the raster’s 25%, 50%, 75% percentile.

**Synopsis**

```
SETOF record ST_Quantile(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] quantiles=NULL);
SETOF record ST_Quantile(raster rast, double precision[] quantiles);
SETOF record ST_Quantile(raster rast, integer nband, double precision[] quantiles);
double precision ST_Quantile(raster rast, double precision quantile);
double precision ST_Quantile(raster rast, boolean exclude_nodata_value, double precision quantile=NULL);
double precision ST_Quantile(raster rast, integer nband, double precision quantile);
double precision ST_Quantile(raster rast, integer nband, boolean exclude_nodata_value, double precision quantile);
double precision ST_Quantile(raster rast, integer nband, double precision quantile);
```

**Description**

Compute quantiles for a raster or raster table coverage in the context of the sample or population. Thus, a value could be examined to be at the raster’s 25%, 50%, 75% percentile.



**Note**

If `exclude_nodata_value` is set to `false`, will also count pixels with no data.

Changed: 3.1.0 Removed `ST_Quantile(table_name, column_name)` variant.

Availability: 2.0.0

Examples

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Example will consider only pixels of band 1 that are not 249 and in named quantiles --

SELECT (pvq).*
FROM (SELECT ST_Quantile(rast, ARRAY[0.25,0.75]) As pvq
 FROM dummy_rast WHERE rid=2) As foo
 ORDER BY (pvq).quantile;

quantile | value
-----+-----
 0.25 | 253
 0.75 | 254

SELECT ST_Quantile(rast, 0.75) As value
 FROM dummy_rast WHERE rid=2;

value

 254
```

```
--real live example. Quantile of all pixels in band 2 intersecting a geometry
SELECT rid, (ST_Quantile(rast,2)).* As pvc
 FROM o_4_boston
 WHERE ST_Intersects(rast,
 ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 892151,224486 892151))',26986)
)
ORDER BY value, quantile,rid
;

rid | quantile | value
-----+-----+-----
 1 | 0 | 0
 2 | 0 | 0
 14 | 0 | 1
 15 | 0 | 2
 14 | 0.25 | 37
 1 | 0.25 | 42
 15 | 0.25 | 47
 2 | 0.25 | 50
 14 | 0.5 | 56
 1 | 0.5 | 64
 15 | 0.5 | 66
 2 | 0.5 | 77
 14 | 0.75 | 81
 15 | 0.75 | 87
 1 | 0.75 | 94
 2 | 0.75 | 106
 14 | 1 | 199
 1 | 1 | 244
 2 | 1 | 255
 15 | 1 | 255
```

See Also

[ST\\_Count](#), [ST\\_SummaryStats](#), [ST\\_SummaryStatsAgg](#), [ST\\_SetBandNoDataValue](#)

### 10.9.5 ST\_SummaryStats

`ST_SummaryStats` — Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed is no band is specified.

#### Synopsis

summarystats **ST\_SummaryStats**(raster rast, boolean exclude\_nodata\_value);  
summarystats **ST\_SummaryStats**(raster rast, integer nband, boolean exclude\_nodata\_value);

#### Description

Returns **summarystats** consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. If no band is specified `nband` defaults to 1.



**Note**  
By default only considers pixel values not equal to the `nodata` value. Set `exclude_nodata_value` to `false` to get count of all pixels.



**Note**  
By default will sample all pixels. To get faster response, set `sample_percent` to lower than 1

Changed: 3.1.0 `ST_SummaryStats(rastertable, rastercolumn, ...)` variants are removed. Use **ST\_SummaryStatsAgg** instead.  
Availability: 2.0.0

#### Example: Single raster tile

```
SELECT rid, band, (stats).*
FROM (SELECT rid, band, ST_SummaryStats(rast, band) As stats
 FROM dummy_rast CROSS JOIN generate_series(1,3) As band
 WHERE rid=2) As foo;
```

| rid | band | count | sum  | mean       | stddev    | min | max |
|-----|------|-------|------|------------|-----------|-----|-----|
| 2   | 1    | 23    | 5821 | 253.086957 | 1.248061  | 250 | 254 |
| 2   | 2    | 25    | 3682 | 147.28     | 59.862188 | 78  | 254 |
| 2   | 3    | 25    | 3290 | 131.6      | 61.647384 | 62  | 254 |

#### Example: Summarize pixels that intersect buildings of interest

This example took 574ms on PostGIS windows 64-bit with all of Boston Buildings and aerial Tiles (tiles each 150x150 pixels ~ 134,000 tiles), ~102,000 building records

```
WITH
-- our features of interest
feat AS (SELECT gid As building_id, geom_26986 As geom FROM buildings AS b
 WHERE gid IN(100, 103,150)
),
-- clip band 2 of raster tiles to boundaries of builds
-- then get stats for these clipped regions
```

```

 b_stats AS
 (SELECT building_id, (stats).*
FROM (SELECT building_id, ST_SummaryStats(ST_Clip(rast,2,geom)) As stats
 FROM aerials.boston
 INNER JOIN feat
 ON ST_Intersects(feat.geom,rast)
) As foo
)
-- finally summarize stats
SELECT building_id, SUM(count) As num_pixels
 , MIN(min) As min_pval
 , MAX(max) As max_pval
 , SUM(mean*count)/SUM(count) As avg_pval
FROM b_stats
WHERE count
> 0
 GROUP BY building_id
 ORDER BY building_id;
building_id | num_pixels | min_pval | max_pval | avg_pval
-----+-----+-----+-----+-----
 100 | 1090 | 1 | 255 | 61.0697247706422
 103 | 655 | 7 | 182 | 70.5038167938931
 150 | 895 | 2 | 252 | 185.642458100559

```

### Example: Raster coverage

```

-- stats for each band --
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band) As stats
 FROM generate_series(1,3) As band) As foo;

```

| band | count   | sum    | mean             | stddev           | min | max |
|------|---------|--------|------------------|------------------|-----|-----|
| 1    | 8450000 | 725799 | 82.7064349112426 | 45.6800222638537 | 0   | 255 |
| 2    | 8450000 | 700487 | 81.4197705325444 | 44.2161184161765 | 0   | 255 |
| 3    | 8450000 | 575943 | 74.682739408284  | 44.2143885481407 | 0   | 255 |

```

-- For a table -- will get better speed if set sampling to less than 100%
-- Here we set to 25% and get a much faster answer
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band,true,0.25) As stats
 FROM generate_series(1,3) As band) As foo;

```

| band | count   | sum    | mean             | stddev           | min | max |
|------|---------|--------|------------------|------------------|-----|-----|
| 1    | 2112500 | 180686 | 82.6890480473373 | 45.6961043857248 | 0   | 255 |
| 2    | 2112500 | 174571 | 81.448503668639  | 44.2252623171821 | 0   | 255 |
| 3    | 2112500 | 144364 | 74.6765884023669 | 44.2014869384578 | 0   | 255 |

### See Also

[summarystats](#), [ST\\_SummaryStatsAgg](#), [ST\\_Count](#), [ST\\_Clip](#)

## 10.9.6 ST\_SummaryStatsAgg

**ST\_SummaryStatsAgg** — Aggregate. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a set of raster. Band 1 is assumed is no band is specified.

## Synopsis

```
summarystats ST_SummaryStatsAgg(raster rast, integer nband, boolean exclude_nodata_value, double precision sample_percent);
summarystats ST_SummaryStatsAgg(raster rast, boolean exclude_nodata_value, double precision sample_percent);
summarystats ST_SummaryStatsAgg(raster rast, integer nband, boolean exclude_nodata_value);
```

## Description

Returns **summarystats** consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. If no band is specified `nband` defaults to 1.



### Note

By default only considers pixel values not equal to the `NODATA` value. Set `exclude_nodata_value` to `False` to get count of all pixels.



### Note

By default will sample all pixels. To get faster response, set `sample_percent` to value between 0 and 1

Availability: 2.2.0

## Examples

```
WITH foo AS (
 SELECT
 rast.rast
 FROM (
 SELECT ST_SetValue(
 ST_SetValue(
 ST_SetValue(
 ST_AddBand(
 ST_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,0)
 , 1, '64BF', 0, 0
)
 , 1, 1, 1, -10
)
 , 1, 5, 4, 0
)
 , 1, 5, 5, 3.14159
) AS rast
) AS rast
 FULL JOIN (
 SELECT generate_series(1, 10) AS id
) AS id
 ON 1 = 1
)
SELECT
 (stats).count,
 round((stats).sum::numeric, 3),
 round((stats).mean::numeric, 3),
 round((stats).stddev::numeric, 3),
 round((stats).min::numeric, 3),
 round((stats).max::numeric, 3)
FROM (
```



```
SELECT
 ST_SummaryStatsAgg(rast, 1, TRUE, 1) AS stats
FROM foo
) bar;
```

| count | round   | round  | round | round   | round |
|-------|---------|--------|-------|---------|-------|
| 20    | -68.584 | -3.429 | 6.571 | -10.000 | 3.142 |

(1 row)

See Also

[summarystats](#), [ST\\_SummaryStats](#), [ST\\_Count](#), [ST\\_Clip](#)

10.9.7 ST\_ValueCount

ST\_ValueCount — Returns a set of records containing a pixel band value and count of the number of pixels in a given band of a raster (or a raster coverage) that have a given set of values. If no band is specified defaults to band 1. By default nodata value pixels are not counted. and all other values in the pixel are output and pixel band values are rounded to the nearest integer.

Synopsis

SETOF record **ST\_ValueCount**(raster rast, integer nband=1, boolean exclude\_nodata\_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);  
SETOF record **ST\_ValueCount**(raster rast, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);  
SETOF record **ST\_ValueCount**(raster rast, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);  
bigint **ST\_ValueCount**(raster rast, double precision searchvalue, double precision roundto=0);  
bigint **ST\_ValueCount**(raster rast, integer nband, boolean exclude\_nodata\_value, double precision searchvalue, double precision roundto=0);  
bigint **ST\_ValueCount**(raster rast, integer nband, double precision searchvalue, double precision roundto=0);  
SETOF record **ST\_ValueCount**(text rastertable, text rastercolumn, integer nband=1, boolean exclude\_nodata\_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);  
SETOF record **ST\_ValueCount**(text rastertable, text rastercolumn, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);  
SETOF record **ST\_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);  
bigint **ST\_ValueCount**(text rastertable, text rastercolumn, integer nband, boolean exclude\_nodata\_value, double precision searchvalue, double precision roundto=0);  
bigint **ST\_ValueCount**(text rastertable, text rastercolumn, double precision searchvalue, double precision roundto=0);  
bigint **ST\_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision searchvalue, double precision roundto=0);

Description

Returns a set of records with columns `value` `count` which contain the pixel band value and count of pixels in the raster tile or raster coverage of selected band.

If no band is specified `nband` defaults to 1. If no `searchvalues` are specified, will return all pixel values found in the raster or raster coverage. If one `searchvalue` is given, will return an integer instead of records denoting the count of pixels having that pixel band value



**Note**  
If `exclude_nodata_value` is set to false, will also count pixels with no data.

Availability: 2.0.0

## Examples

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Example will count only pixels of band 1 that are not 249. --
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast) As pvc
 FROM dummy_rast WHERE rid=2) As foo
 ORDER BY (pvc).value;
```

| value | count |
|-------|-------|
| 250   | 2     |
| 251   | 1     |
| 252   | 2     |
| 253   | 6     |
| 254   | 12    |

```
-- Example will count all pixels of band 1 including 249 --
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,1,false) As pvc
 FROM dummy_rast WHERE rid=2) As foo
 ORDER BY (pvc).value;
```

| value | count |
|-------|-------|
| 249   | 2     |
| 250   | 2     |
| 251   | 1     |
| 252   | 2     |
| 253   | 6     |
| 254   | 12    |

```
-- Example will count only non-nodata value pixels of band 2
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,2) As pvc
 FROM dummy_rast WHERE rid=2) As foo
 ORDER BY (pvc).value;
```

| value | count |
|-------|-------|
| 78    | 1     |
| 79    | 1     |
| 88    | 1     |
| 89    | 1     |
| 96    | 1     |
| 97    | 1     |
| 98    | 1     |
| 99    | 2     |
| 112   | 2     |

```
:
```

```
--real live example. Count all the pixels in an aerial raster tile band 2 intersecting a ←
 geometry
-- and return only the pixel band values that have a count
> 500
```

```
SELECT (pvc).value, SUM((pvc).count) As total
FROM (SELECT ST_ValueCount(rast,2) As pvc
 FROM o_4_boston
```

```

 WHERE ST_Intersects(rast,
 ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706
 892151,224486 892151))',26986)
) As foo
 GROUP BY (pvc).value
 HAVING SUM((pvc).count)
> 500
 ORDER BY (pvc).value;

value | total
-----+-----
 51 | 502
 54 | 521

```

```

-- Just return count of pixels in each raster tile that have value of 100 of tiles that
-- intersect a specific geometry --
SELECT rid, ST_ValueCount(rast,2,100) As count
 FROM o_4_boston
 WHERE ST_Intersects(rast,
 ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706
 892151,224486 892151))',26986)
) ;

rid | count
-----+-----
 1 | 56
 2 | 95
 14 | 37
 15 | 64

```

## See Also

[ST\\_Count](#), [ST\\_SetBandNoDataValue](#)

## 10.10 Raster Inputs

### 10.10.1 ST\_RastFromWKB

**ST\_RastFromWKB** — Return a raster value from a Well-Known Binary (WKB) raster.

#### Synopsis

```
raster ST_RastFromWKB(bytea wkb);
```

#### Description

Given a Well-Known Binary (WKB) raster, return a raster.

Availability: 2.5.0







## Using PostgreSQL Large Object Support to export raster

One way to export raster into another format is using [PostgreSQL large object export functions](#). We'll repeat the prior example but also exporting. Note for this you'll need to have super user access to db since it uses server side lo functions. It will also export to path on server network. If you need export locally, use the psql equivalent lo\_ functions which export to the local file system instead of the server file system.

```
DROP TABLE IF EXISTS tmp_out ;

CREATE TABLE tmp_out AS
SELECT lo_from_bytea(0,
 ST_AsGDALRaster(ST_Union(rast), 'JPEG', ARRAY['QUALITY=50']))
) AS loid
FROM dummy_rast
WHERE rast && ST_MakeEnvelope(10, 10, 11, 11);

SELECT lo_export(loid, '/tmp/dummy.jpg')
FROM tmp_out;

SELECT lo_unlink(loid)
FROM tmp_out;
```

## GTIFF Output Examples

```
SELECT ST_AsGDALRaster(rast, 'GTiff') As rastjpg
FROM dummy_rast WHERE rid=2;

-- Out GeoTiff with jpeg compression, 90% quality
SELECT ST_AsGDALRaster(rast, 'GTiff',
 ARRAY['COMPRESS=JPEG', 'JPEG_QUALITY=90'],
 4269) As rasttiff
FROM dummy_rast WHERE rid=2;
```

## See Also

Section [9.3](#), [ST\\_GDALDrivers](#), [ST\\_SRID](#)

### 10.11.4 ST\_AsJPEG

**ST\_AsJPEG** — Return the raster tile selected bands as a single Joint Photographic Exports Group (JPEG) image (byte array). If no band is specified and 1 or more than 3 bands, then only the first band is used. If only 3 bands then all 3 bands are used and mapped to RGB.

## Synopsis

```
bytea ST_AsJPEG(raster rast, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer nband, integer quality);
bytea ST_AsJPEG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, integer quality);
```

## Description

Returns the selected bands of the raster as a single Joint Photographic Exports Group Image (JPEG). Use **ST\_AsGDALRaster** if you need to export as less common raster types. If no band is specified and 1 or more than 3 bands, then only the first band is used. If 3 bands then all 3 bands are used. There are many variants of the function with many options. These are itemized below:

- `nband` is for single band exports.
- `nbands` is an array of bands to export (note that max is 3 for JPEG) and the order of the bands is RGB. e.g `ARRAY[3,2,1]` means map band 3 to Red, band 2 to green and band 1 to blue
- `quality` number from 0 to 100. The higher the number the crisper the image.
- `options` text Array of GDAL options as defined for JPEG (look at `create_options` for JPEG **ST\_GDALDrivers**). For JPEG valid ones are `PROGRESSIVE ON` or `OFF` and `QUALITY` a range from 0 to 100 and default to 75. Refer to **GDAL Raster format options** for more details.

Availability: 2.0.0 - requires GDAL >= 1.6.0.

## Examples: Output

```
-- output first 3 bands 75% quality
SELECT ST_AsJPEG(rast) As rastjpg
 FROM dummy_rast WHERE rid=2;

-- output only first band as 90% quality
SELECT ST_AsJPEG(rast,1,90) As rastjpg
 FROM dummy_rast WHERE rid=2;

-- output first 3 bands (but make band 2 Red, band 1 green, and band 3 blue, progressive ↵
and 90% quality
SELECT ST_AsJPEG(rast,ARRAY[2,1,3],ARRAY['QUALITY=90','PROGRESSIVE=ON']) As rastjpg
 FROM dummy_rast WHERE rid=2;
```

## See Also

Section 9.3, **ST\_GDALDrivers**, **ST\_AsGDALRaster**, **ST\_AsPNG**, **ST\_AsTIFF**

## 10.11.5 ST\_AsPNG

**ST\_AsPNG** — Return the raster tile selected bands as a single portable network graphics (PNG) image (byte array). If 1, 3, or 4 bands in raster and no bands are specified, then all bands are used. If more 2 or more than 4 bands and no bands specified, then only band 1 is used. Bands are mapped to RGB or RGBA space.

## Synopsis

```
bytea ST_AsPNG(raster rast, text[] options=NULL);
bytea ST_AsPNG(raster rast, integer nband, integer compression);
bytea ST_AsPNG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsPNG(raster rast, integer[] nbands, integer compression);
bytea ST_AsPNG(raster rast, integer[] nbands, text[] options=NULL);
```



## Description

Returns the selected bands of the raster as a single Portable Network Graphics Image (PNG). Use [ST\\_AsGDALRaster](#) if you need to export as less common raster types. If no band is specified, then the first 3 bands are exported. There are many variants of the function with many options. If no `srid` is specified then the `srid` of the raster is used. These are itemized below:

- `nband` is for single band exports.
- `nbands` is an array of bands to export (note that max is 4 for PNG) and the order of the bands is RGBA. e.g. `ARRAY[3,2,1]` means map band 3 to Red, band 2 to green and band 1 to blue
- `compression` number from 1 to 9. The higher the number the greater the compression.
- `options` text Array of GDAL options as defined for PNG (look at `create_options` for PNG of [ST\\_GDALDrivers](#)). For PNG valid one is only `ZLEVEL` (amount of time to spend on compression -- default 6) e.g. `ARRAY['ZLEVEL=9']`. `WORLDFILE` is not allowed since the function would have to output two outputs. Refer to [GDAL Raster format options](#) for more details.

Availability: 2.0.0 - requires GDAL >= 1.6.0.

## Examples

```
SELECT ST_AsPNG(rast) As rastpng
FROM dummy_rast WHERE rid=2;

-- export the first 3 bands and map band 3 to Red, band 1 to Green, band 2 to blue
SELECT ST_AsPNG(rast, ARRAY[3,1,2]) As rastpng
FROM dummy_rast WHERE rid=2;
```

## See Also

[ST\\_AsGDALRaster](#), [ST\\_ColorMap](#), [ST\\_GDALDrivers](#), [Section 9.3](#)

### 10.11.6 ST\_AsTIFF

**ST\_AsTIFF** — Return the raster selected bands as a single TIFF image (byte array). If no band is specified or any of specified bands does not exist in the raster, then will try to use all bands.

## Synopsis

```
bytea ST_AsTIFF(raster rast, text[] options='', integer srid=sameassource);
bytea ST_AsTIFF(raster rast, text compression='', integer srid=sameassource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text compression='', integer srid=sameassource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text[] options, integer srid=sameassource);
```

## Description

Returns the selected bands of the raster as a single Tagged Image File Format (TIFF). If no band is specified, will try to use all bands. This is a wrapper around [ST\\_AsGDALRaster](#). Use [ST\\_AsGDALRaster](#) if you need to export as less common raster types. There are many variants of the function with many options. If no spatial reference SRS text is present, the spatial reference of the raster is used. These are itemized below:

- `nbands` is an array of bands to export (note that max is 3 for PNG) and the order of the bands is RGB. e.g. `ARRAY[3,2,1]` means map band 3 to Red, band 2 to green and band 1 to blue

- `compression` Compression expression -- JPEG90 (or some other percent), LZW, JPEG, DEFLATE9.
- `options` text Array of GDAL create options as defined for GTiff (look at `create_options` for GTiff of [ST\\_GDALDrivers](#)). or refer to [GDAL Raster format options](#) for more details.
- `srid` srid of spatial\_ref\_sys of the raster. This is used to populate the georeference information

Availability: 2.0.0 - requires GDAL >= 1.6.0.

#### Examples: Use jpeg compression 90%

```
SELECT ST_AsTIFF(rast, 'JPEG90') As rasttiff
FROM dummy_rast WHERE rid=2;
```

#### See Also

[ST\\_GDALDrivers](#), [ST\\_AsGDALRaster](#), [ST\\_SRID](#)

## 10.12 Raster Processing: Map Algebra

### 10.12.1 ST\_Clip

**ST\_Clip** — Returns the raster clipped by the input geometry. If band number not is specified, all bands are processed. If `crop` is not specified or TRUE, the output raster is cropped.

#### Synopsis

```
raster ST_Clip(raster rast, integer[] nband, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE);
raster ST_Clip(raster rast, integer nband, geometry geom, double precision nodataval, boolean crop=TRUE);
raster ST_Clip(raster rast, integer nband, geometry geom, boolean crop);
raster ST_Clip(raster rast, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE);
raster ST_Clip(raster rast, geometry geom, double precision nodataval, boolean crop=TRUE);
raster ST_Clip(raster rast, geometry geom, boolean crop);
```

#### Description

Returns a raster that is clipped by the input geometry `geom`. If band index is not specified, all bands are processed.

Rasters resulting from **ST\_Clip** must have a nodata value assigned for areas clipped, one for each band. If none are provided and the input raster do not have a nodata value defined, nodata values of the resulting raster are set to `ST_MinPossibleValue(ST_BandPixelType(rast), band)`. When the number of nodata value in the array is smaller than the number of band, the last one in the array is used for the remaining bands. If the number of nodata value is greater than the number of band, the extra nodata values are ignored. All variants accepting an array of nodata values also accept a single value which will be assigned to each band.

If `crop` is not specified, true is assumed meaning the output raster is cropped to the intersection of the `geom` and `rast` extents. If `crop` is set to false, the new raster gets the same extent as `rast`.

Availability: 2.0.0

Enhanced: 2.1.0 Rewritten in C

Examples here use Massachusetts aerial data available on MassGIS site [MassGIS Aerial Orthos](#). Coordinates are in Massachusetts State Plane Meters.

Examples: 1 band clipping

```
-- Clip the first band of an aerial tile by a 20 meter buffer.
SELECT ST_Clip(rast, 1,
 ST_Buffer(ST_Centroid(ST_Envelope(rast)),20)
) from aerials.boston
WHERE rid = 4;
```

```
-- Demonstrate effect of crop on final dimensions of raster
-- Note how final extent is clipped to that of the geometry
-- if crop = true
SELECT ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, true))) As xmax_w_trim,
 ST_XMax(clipper) As xmax_clipper,
 ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, false))) As xmax_wo_trim,
 ST_XMax(ST_Envelope(rast)) As xmax_rast_orig
FROM (SELECT rast, ST_Buffer(ST_Centroid(ST_Envelope(rast)),6) As clipper
 FROM aerials.boston
 WHERE rid = 6) As foo;
```

| xmax_w_trim      | xmax_clipper     | xmax_wo_trim     | xmax_rast_orig   |
|------------------|------------------|------------------|------------------|
| 230657.436173996 | 230657.436173996 | 230666.436173996 | 230666.436173996 |



Examples: 1 band clipping with no crop and add back other bands unchanged

```
-- Same example as before, but we need to set crop to false to be able to use ST_AddBand
-- because ST_AddBand requires all bands be the same Width and height
SELECT ST_AddBand(ST_Clip(rast, 1,
 ST_Buffer(ST_Centroid(ST_Envelope(rast)),20),false
), ARRAY[ST_Band(rast,2),ST_Band(rast,3)]) from aerials.boston
WHERE rid = 6;
```

*Full raster tile before clipping**After Clipping - surreal***Examples: Clip all bands**

```
-- Clip all bands of an aerial tile by a 20 meter buffer.
-- Only difference is we don't specify a specific band to clip
-- so all bands are clipped
SELECT ST_Clip(rast,
 ST_Buffer(ST_Centroid(ST_Envelope(rast)), 20),
 false
) from aerials.boston
WHERE rid = 4;
```

*Full raster tile before clipping**After Clipping***See Also**

[ST\\_AddBand](#), [ST\\_MapAlgebra](#) (callback function version), [ST\\_Intersection](#)

### 10.12.2 ST\_ColorMap

**ST\_ColorMap** — Creates a new raster of up to four 8BUI bands (grayscale, RGB, RGBA) from the source raster and a specified band. Band 1 is assumed if not specified.

#### Synopsis

```
raster ST_ColorMap(raster rast, integer nband=1, text colormap=grayscale, text method=INTERPOLATE);
```

```
raster ST_ColorMap(raster rast, text colormap, text method=INTERPOLATE);
```

#### Description

Apply a `colormap` to the band at `nband` of `rast` resulting a new raster comprised of up to four 8BUI bands. The number of 8BUI bands in the new raster is determined by the number of color components defined in `colormap`.

If `nband` is not specified, then band 1 is assumed.

`colormap` can be a keyword of a pre-defined colormap or a set of lines defining the value and the color components.

Valid pre-defined `colormap` keyword:

- `grayscale` or `greyscale` for a one 8BUI band raster of shades of gray.
- `pseudocolor` for a four 8BUI (RGBA) band raster with colors going from blue to green to red.
- `fire` for a four 8BUI (RGBA) band raster with colors going from black to red to pale yellow.
- `bluered` for a four 8BUI (RGBA) band raster with colors going from blue to pale white to red.

Users can pass a set of entries (one per line) to `colormap` to specify custom colormaps. Each entry generally consists of five values: the pixel value and corresponding Red, Green, Blue, Alpha components (color components between 0 and 255). Percent values can be used instead of pixel values where 0% and 100% are the minimum and maximum values found in the raster band. Values can be separated with commas (','), tabs, colons (':') and/or spaces. The pixel value can be set to *nv*, *null* or *nodata* for the NODATA value. An example is provided below.

```
5 0 0 0 255
4 100:50 55 255
1 150,100 150 255
0% 255 255 255 255
nv 0 0 0 0
```

The syntax of `colormap` is similar to that of the color-relief mode of GDAL [gdaldem](#).

Valid keywords for `method`:

- `INTERPOLATE` to use linear interpolation to smoothly blend the colors between the given pixel values
- `EXACT` to strictly match only those pixels values found in the colormap. Pixels whose value does not match a colormap entry will be set to 0 0 0 0 (RGBA)
- `NEAREST` to use the colormap entry whose value is closest to the pixel value



#### Note

A great reference for colormaps is [ColorBrewer](#).

---



**Warning**

The resulting bands of new raster will have no NODATA value set. Use `ST_SetBandNoDataValue` to set a NODATA value if one is needed.

Availability: 2.1.0

**Examples**

This is a junk table to play with

```
-- setup test raster table --
DROP TABLE IF EXISTS funky_shapes;
CREATE TABLE funky_shapes(rast raster);

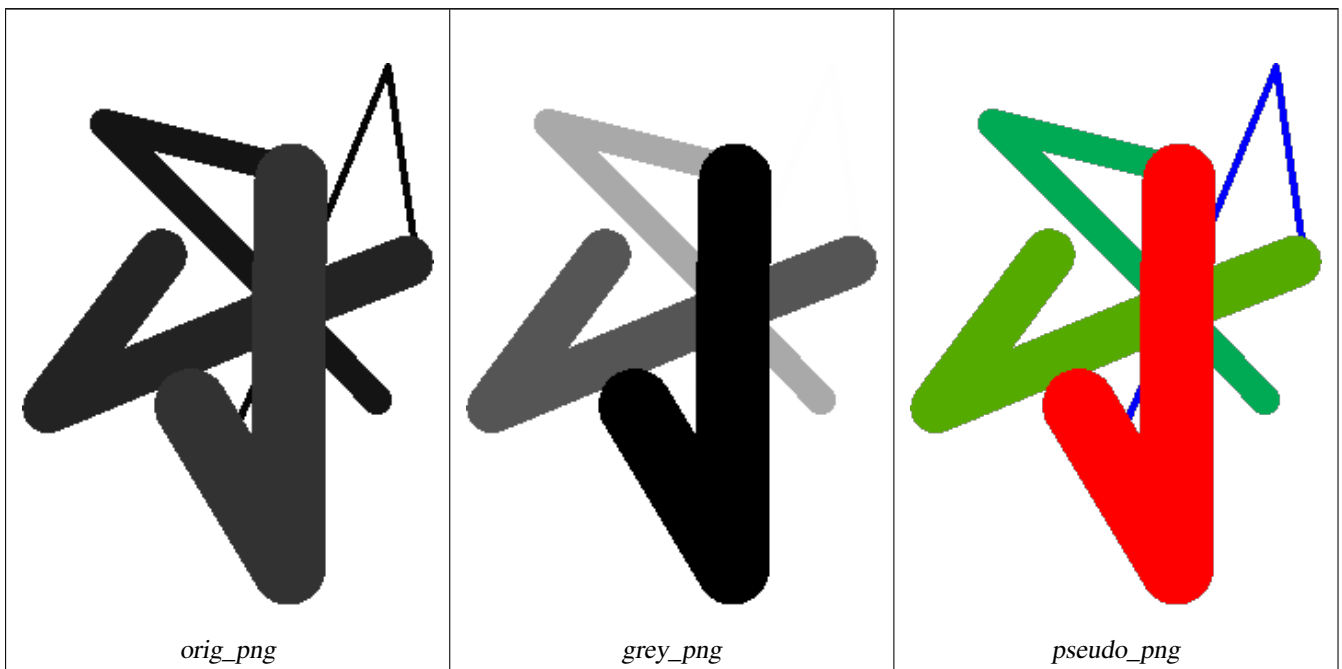
INSERT INTO funky_shapes(rast)
WITH ref AS (
 SELECT ST_MakeEmptyRaster(200, 200, 0, 200, 1, -1, 0, 0) AS rast
)
SELECT
 ST_Union(rast)
FROM (
 SELECT
 ST_AsRaster(
 ST_Rotate(
 ST_Buffer(
 ST_GeomFromText('LINESTRING(0 2,50 50,150 150,125 50)'),
 i*2
),
 pi() * i * 0.125, ST_Point(50,50)
),
 ref.rast, '8BUI'::text, i * 5
) AS rast
 FROM ref
 CROSS JOIN generate_series(1, 10, 3) AS i
) AS shapes;
```

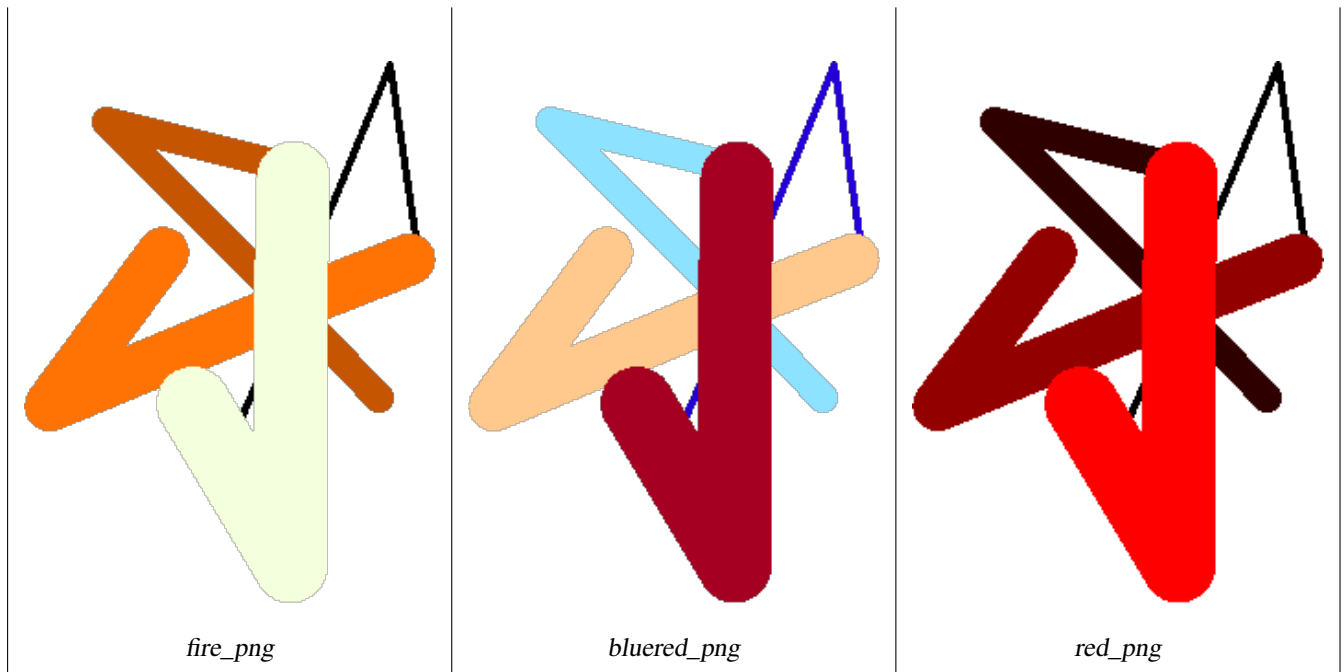
```
SELECT
 ST_NumBands(rast) As n_orig,
 ST_NumBands(ST_ColorMap(rast,1, 'greyscale')) As ngrey,
 ST_NumBands(ST_ColorMap(rast,1, 'pseudocolor')) As npseudo,
 ST_NumBands(ST_ColorMap(rast,1, 'fire')) As nfire,
 ST_NumBands(ST_ColorMap(rast,1, 'bluered')) As nbluered,
 ST_NumBands(ST_ColorMap(rast,1, '
100% 255 0 0
80% 160 0 0
50% 130 0 0
30% 30 0 0
20% 60 0 0
0% 0 0 0
nv 255 255 255
')) As nred
FROM funky_shapes;
```

| n_orig | ngrey | npseudo | nfire | nbluered | nred |
|--------|-------|---------|-------|----------|------|
| 1      | 1     | 4       | 4     | 4        | 3    |

**Examples: Compare different color map looks using ST\_AsPNG**

```
SELECT
 ST_AsPNG(rast) As orig_png,
 ST_AsPNG(ST_ColorMap(rast,1,'greyscale')) As grey_png,
 ST_AsPNG(ST_ColorMap(rast,1, 'pseudocolor')) As pseudo_png,
 ST_AsPNG(ST_ColorMap(rast,1, 'nfire')) As fire_png,
 ST_AsPNG(ST_ColorMap(rast,1, 'bluered')) As bluered_png,
 ST_AsPNG(ST_ColorMap(rast,1, '
100% 255 0 0
80% 160 0 0
50% 130 0 0
30% 30 0 0
20% 60 0 0
0% 0 0 0
nv 255 255 255
')) As red_png
FROM funky_shapes;
```





### See Also

[ST\\_AsPNG](#), [ST\\_AsRaster](#) [ST\\_MapAlgebra](#) (callback function version), [ST\\_Grayscale](#) [ST\\_NumBands](#), [ST\\_Reclass](#), [ST\\_SetBandNoDataValue](#), [ST\\_Union](#)

### 10.12.3 ST\_Grayscale

**ST\_Grayscale** — Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue

#### Synopsis

- (1) raster **ST\_Grayscale**(raster rast, integer redband=1, integer greenband=2, integer blueband=3, text extenttype=INTERSECTION);
- (2) raster **ST\_Grayscale**(rastbandarg[] rastbandargset, text extenttype=INTERSECTION);

#### Description

Create a raster with one 8BUI band given three input bands (from one or more rasters). Any input band whose pixel type is not 8BUI will be reclassified using [ST\\_Reclass](#).



#### Note

This function is not like [ST\\_ColorMap](#) with the `grayscale` keyword as [ST\\_ColorMap](#) operates on only one band while this function expects three bands for RGB. This function applies the following equation for converting RGB to Grayscale:  $0.2989 * RED + 0.5870 * GREEN + 0.1140 * BLUE$

Availability: 2.5.0



**Examples: Variant 1**

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
 SELECT ST_AddBand(
 ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
 '/tmp/apple.png'::text,
 NULL::int[]
) AS rast
)
SELECT
 ST_AsPNG(rast) AS original_png,
 ST_AsPNG(ST_Grayscale(rast)) AS grayscale_png
FROM apple;
```

*original\_png**grayscale\_png***Examples: Variant 2**

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
 SELECT ST_AddBand(
 ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
 '/tmp/apple.png'::text,
 NULL::int[]
) AS rast
)
SELECT
 ST_AsPNG(rast) AS original_png,
 ST_AsPNG(ST_Grayscale(
 ARRAY[
 ROW(rast, 1)::rastbandarg, -- red
 ROW(rast, 2)::rastbandarg, -- green
 ROW(rast, 3)::rastbandarg, -- blue
]::rastbandarg[]
)) AS grayscale_png
FROM apple;
```

**See Also**

[ST\\_AsPNG](#), [ST\\_Reclass](#), [ST\\_ColorMap](#)

**10.12.4 ST\_Intersection**

**ST\_Intersection** — Returns a raster or a set of geometry-pixelvalue pairs representing the shared portion of two rasters or the geometrical intersection of a vectorization of the raster and a geometry.

**Synopsis**

```
setof geomval ST_Intersection(geometry geom, raster rast, integer band_num=1);
setof geomval ST_Intersection(raster rast, geometry geom);
setof geomval ST_Intersection(raster rast, integer band, geometry geom);
raster ST_Intersection(raster rast1, raster rast2, double precision[] nodataval);
raster ST_Intersection(raster rast1, raster rast2, text returnband, double precision[] nodataval);
raster ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval);
raster ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval);
```

**Description**

Returns a raster or a set of geometry-pixelvalue pairs representing the shared portion of two rasters or the geometrical intersection of a vectorization of the raster and a geometry.

The first three variants, returning a setof geomval, works in vector space. The raster is first vectorized (using [ST\\_DumpAsPolygons](#)) into a set of geomval rows and those rows are then intersected with the geometry using the [ST\\_Intersection](#) (geometry, geometry) PostGIS function. Geometries intersecting only with a nodata value area of a raster returns an empty geometry. They are normally excluded from the results by the proper usage of [ST\\_Intersects](#) in the WHERE clause.

You can access the geometry and the value parts of the resulting set of geomval by surrounding them with parenthesis and adding '.geom' or '.val' at the end of the expression. e.g. (ST\_Intersection(rast, geom)).geom

The other variants, returning a raster, works in raster space. They are using the two rasters version of ST\_MapAlgebraExpr to perform the intersection.

The extent of the resulting raster corresponds to the geometrical intersection of the two raster extents. The resulting raster includes 'BAND1', 'BAND2' or 'BOTH' bands, following what is passed as the `returnband` parameter. Nodata value areas present in any band results in nodata value areas in every bands of the result. In other words, any pixel intersecting with a nodata value pixel becomes a nodata value pixel in the result.

Rasters resulting from ST\_Intersection must have a nodata value assigned for areas not intersecting. You can define or replace the nodata value for any resulting band by providing a `nodataval[]` array of one or two nodata values depending if you request 'BAND1', 'BAND2' or 'BOTH' bands. The first value in the array replace the nodata value in the first band and the second value replace the nodata value in the second band. If one input band do not have a nodata value defined and none are provided as an array, one is chosen using the ST\_MinPossibleValue function. All variant accepting an array of nodata value can also accept a single value which will be assigned to each requested band.

In all variants, if no band number is specified band 1 is assumed. If you need an intersection between a raster and geometry that returns a raster, refer to [ST\\_Clip](#).

**Note**

To get more control on the resulting extent or on what to return when encountering a nodata value, use the two rasters version of [ST\\_MapAlgebraExpr](#).



**Note**  
To compute the intersection of a raster band with a geometry in raster space, use **ST\_Clip**. ST\_Clip works on multiple bands rasters and does not return a band corresponding to the rasterized geometry.



**Note**  
ST\_Intersection should be used in conjunction with **ST\_Intersects** and an index on the raster column and/or the geometry column.

Enhanced: 2.0.0 - Intersection in the raster space was introduced. In earlier pre-2.0.0 versions, only intersection performed in vector space were supported.

**Examples: Geometry, Raster -- resulting in geometry vals**

```
SELECT
 foo.rid,
 foo.gid,
 ST_AsText((foo.geomval).geom) As geomwkt,
 (foo.geomval).val
FROM (
 SELECT
 A.rid,
 g.gid,
 ST_Intersection(A.rast, g.geom) As geomval
 FROM dummy_rast AS A
 CROSS JOIN (
 VALUES
 (1, ST_Point(3427928, 5793243.85)),
 (2, ST_GeomFromText('LINESTRING(3427927.85 5793243.75,3427927.8 5793243.75,3427927.8 5793243.8)')),
 (3, ST_GeomFromText('LINESTRING(1 2, 3 4)'))
) As g(gid,geom)
 WHERE A.rid = 2
) As foo;
```

| rid | gid | geomwkt                                                         | val |
|-----|-----|-----------------------------------------------------------------|-----|
| 2   | 1   | POINT(3427928 5793243.85)                                       | 249 |
| 2   | 1   | POINT(3427928 5793243.85)                                       | 253 |
| 2   | 2   | POINT(3427927.85 5793243.75)                                    | 254 |
| 2   | 2   | POINT(3427927.8 5793243.8)                                      | 251 |
| 2   | 2   | POINT(3427927.8 5793243.8)                                      | 253 |
| 2   | 2   | LINESTRING(3427927.8 5793243.75,3427927.8 5793243.8)            | 252 |
| 2   | 2   | MULTILINESTRING((3427927.8 5793243.8,3427927.8 5793243.75),...) | 250 |
| 2   | 3   | GEOMETRYCOLLECTION EMPTY                                        |     |

**See Also**

geomval, ST\_Intersects, ST\_MapAlgebraExpr, ST\_Clip, ST\_AsText

**10.12.5 ST\_MapAlgebra (callback function version)**

ST\_MapAlgebra (callback function version) — Callback function version - Returns a one-band raster given one or more input rasters, band indexes and one user-specified callback function.

## Synopsis

```
raster ST_MapAlgebra(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=INTERSECTION,
raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer[] nband, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=FIRST,
raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=FIRST,
raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast1, integer nband1, raster rast2, integer nband2, regprocedure callbackfunc, text pixeltype=NULL,
text extenttype=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC user-
args=NULL);
raster ST_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, float8[] mask, boolean weighted, text pixel-
type=NULL, text extenttype=INTERSECTION, raster customextent=NULL, text[] VARIADIC userargs=NULL);
```

## Description

Returns a one-band raster given one or more input rasters, band indexes and one user-specified callback function.

**rast,rast1,rast2, rastbandargset** Rasters on which the map algebra process is evaluated.

`rastbandargset` allows the use of a map algebra operation on many rasters and/or many bands. See example Variant 1.

**nband, nband1, nband2** Band numbers of the raster to be evaluated. `nband` can be an integer or integer[] denoting the bands. `nband1` is band on `rast1` and `nband2` is band on `rast2` for the 2 raster/2band case.

**callbackfunc** The `callbackfunc` parameter must be the name and signature of an SQL or PL/pgSQL function, cast to a regprocedure. An example PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION sample_callbackfunc(value double precision[][][], position integer[][], VARIADIC userargs text[])
RETURNS double precision
AS $$
BEGIN
 RETURN 0;
END;
$$ LANGUAGE 'plpgsql' IMMUTABLE;
```

The `callbackfunc` must have three arguments: a 3-dimension double precision array, a 2-dimension integer array and a variadic 1-dimension text array. The first argument `value` is the set of values (as double precision) from all input rasters. The three dimensions (where indexes are 1-based) are: raster #, row y, column x. The second argument `position` is the set of pixel positions from the output raster and input rasters. The outer dimension (where indexes are 0-based) is the raster #. The position at outer dimension index 0 is the output raster's pixel position. For each outer dimension, there are two elements in the inner dimension for X and Y. The third argument `userargs` is for passing through any user-specified arguments.

Passing a regprocedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'sample_callbackfunc(double precision[], integer[], text[])':regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

**mask** An n-dimensional array (matrix) of numbers used to filter what cells get passed to map algebra call-back function. 0 means a neighbor cell value should be treated as no-data and 1 means value should be treated as data. If weight is set to true, then the values, are used as multipliers to multiple the pixel value of that value in the neighborhood position.

**weighted** boolean (true/false) to denote if a mask value should be weighted (multiplied by original value) or not (only applies to proto that takes a mask).

**pixeltype** If `pixeltype` is passed in, the one band of the new raster will be of that `pixeltype`. If `pixeltype` is passed NULL or left out, the new raster band will have the same `pixeltype` as the specified band of the first raster (for extent types: INTERSECTION, UNION, FIRST, CUSTOM) or the specified band of the appropriate raster (for extent types: SECOND, LAST). If in doubt, always specify `pixeltype`.

The resulting pixel type of the output raster must be one listed in [ST\\_BandPixelType](#) or left out or set to NULL.

**extenttype** Possible values are INTERSECTION (default), UNION, FIRST (default for one raster variants), SECOND, LAST, CUSTOM.

**customextent** If `extenttype` is CUSTOM, a raster must be provided for `customextent`. See example 4 of Variant 1.

**distance<sub>x</sub>** The distance in pixels from the reference cell in x direction. So width of resulting matrix would be  $2 \times \text{distance}_x + 1$ . If not specified only the reference cell is considered (neighborhood of 0).

**distance<sub>y</sub>** The distance in pixels from reference cell in y direction. Height of resulting matrix would be  $2 \times \text{distance}_y + 1$ . If not specified only the reference cell is considered (neighborhood of 0).

**userargs** The third argument to the `callbackfunc` is a variadic text array. All trailing text arguments are passed through to the specified `callbackfunc`, and are contained in the `userargs` argument.



#### Note

For more information about the VARIADIC keyword, please refer to the PostgreSQL documentation and the "SQL Functions with Variable Numbers of Arguments" section of [Query Language \(SQL\) Functions](#).



#### Note

The `text[]` argument to the `callbackfunc` is required, regardless of whether you choose to pass any arguments to the callback function for processing or not.

Variant 1 accepts an array of `rastbandarg` allowing the use of a map algebra operation on many rasters and/or many bands. See example Variant 1.

Variants 2 and 3 operate upon one or more bands of one raster. See example Variant 2 and 3.

Variant 4 operate upon two rasters with one band per raster. See example Variant 4.

Availability: 2.2.0: Ability to add a mask

Availability: 2.1.0

## Examples: Variant 1

### One raster, one band

```
WITH foo AS (
 SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', ↵
 1, 0) AS rast
)
SELECT
 ST_MapAlgebra(
 ARRAY[ROW(rast, 1)]::rastbandarg[],
 'sample_callbackfunc(double precision[], int[], text[])::regprocedure
) AS rast
FROM foo
```

### One raster, several bands

```

WITH foo AS (
 SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
 ST_MapAlgebra(
 ARRAY[ROW(rast, 3), ROW(rast, 1), ROW(rast, 3), ROW(rast, 2)]::rastbandarg[],
 'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
) AS rast
FROM foo

```

### Several rasters, several bands

```

WITH foo AS (
 SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION ALL
 SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
 ST_MapAlgebra(
 ARRAY[ROW(t1.rast, 3), ROW(t2.rast, 1), ROW(t2.rast, 3), ROW(t1.rast, 2)]::rastbandarg[],
 'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
 AND t2.rid = 2

```

Complete example of tiles of a coverage with neighborhood. This query only works with PostgreSQL 9.1 or higher.

```

WITH foo AS (
 SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', 1, 0) AS rast UNION ALL
 SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0) AS rast UNION ALL
 SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0) AS rast UNION ALL

 SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10, 0) AS rast UNION ALL
 SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20, 0) AS rast UNION ALL
 SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30, 0) AS rast UNION ALL

 SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100, 0) AS rast UNION ALL
 SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200, 0) AS rast UNION ALL
 SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300, 0) AS rast
)
SELECT
 t1.rid,
 ST_MapAlgebra(
 ARRAY[ROW(ST_Union(t2.rast), 1)]::rastbandarg[],
 'sample_callbackfunc(double precision[], int[], text[])'::regprocedure,
 '32BUI',
)

```

```

 'CUSTOM', t1.rast,
 1, 1
) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 4
 AND t2.rid BETWEEN 0 AND 8
 AND ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rid, t1.rast

```

Example like the prior one for tiles of a coverage with neighborhood but works with PostgreSQL 9.0.

```

WITH src AS (
 SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', 1, 0) AS rast UNION ALL
 SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0) AS rast UNION ALL
 SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0) AS rast UNION ALL

 SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10, 0) AS rast UNION ALL
 SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20, 0) AS rast UNION ALL
 SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30, 0) AS rast UNION ALL

 SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100, 0) AS rast UNION ALL
 SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200, 0) AS rast UNION ALL
 SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300, 0) AS rast
)
WITH foo AS (
 SELECT
 t1.rid,
 ST_Union(t2.rast) AS rast
 FROM src t1
 JOIN src t2
 ON ST_Intersects(t1.rast, t2.rast)
 AND t2.rid BETWEEN 0 AND 8
 WHERE t1.rid = 4
 GROUP BY t1.rid
), bar AS (
 SELECT
 t1.rid,
 ST_MapAlgebra(
 ARRAY[ROW(t2.rast, 1)::rastbandarg[],
 'raster_nmapalgebra_test(double precision[], int[], text[])::regprocedure,
 '32BUI',
 'CUSTOM', t1.rast,
 1, 1
] AS rast
 FROM src t1
 JOIN foo t2
 ON t1.rid = t2.rid
)
SELECT
 rid,
 (ST_Metadatas(rast)),
 (ST_BandMetadatas(rast, 1)),

```

```
ST_Value(rast, 1, 1, 1)
FROM bar;
```

### Examples: Variants 2 and 3

#### One raster, several bands

```
WITH foo AS (
 SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↵
 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
 ST_MapAlgebra(
 rast, ARRAY[3, 1, 3, 2]::integer[],
 'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
) AS rast
FROM foo
```

#### One raster, one band

```
WITH foo AS (
 SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↵
 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
 ST_MapAlgebra(
 rast, 2,
 'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
) AS rast
FROM foo
```

### Examples: Variant 4

#### Two rasters, two bands

```
WITH foo AS (
 SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↵
 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION ↵
 ALL
 SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ↵
 0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
 ST_MapAlgebra(
 t1.rast, 2,
 t2.rast, 1,
 'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
 AND t2.rid = 2
```

### Examples: Using Masks

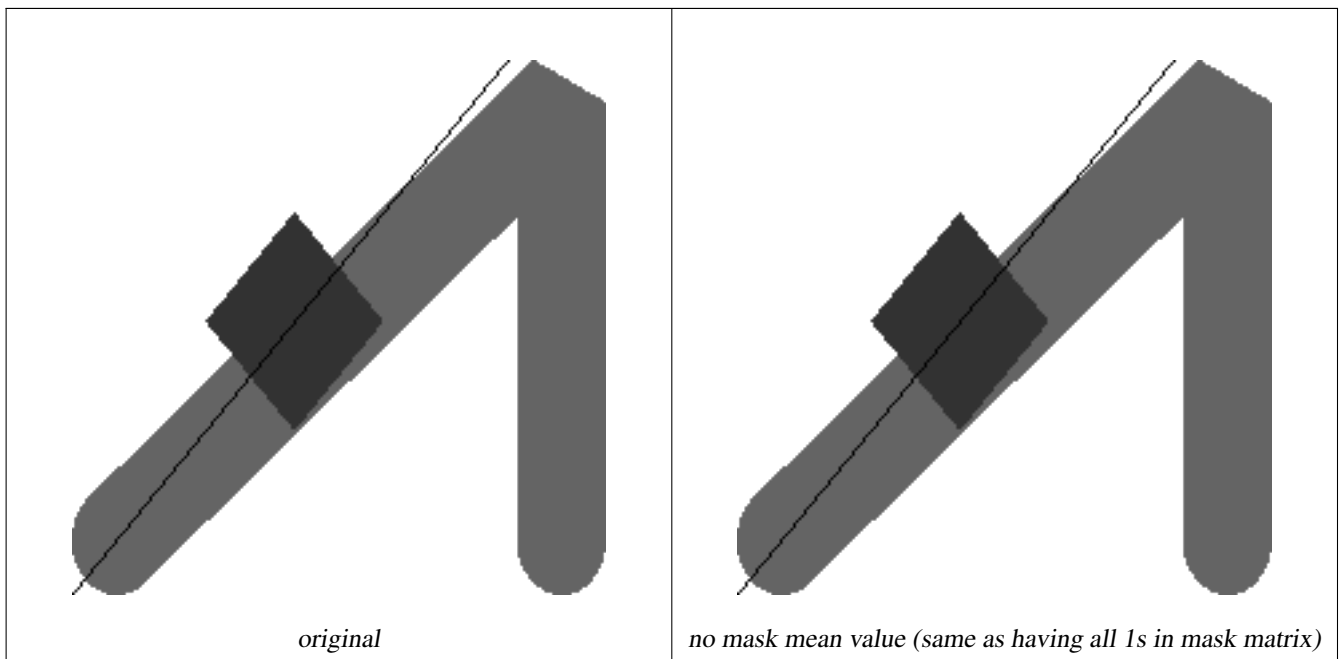


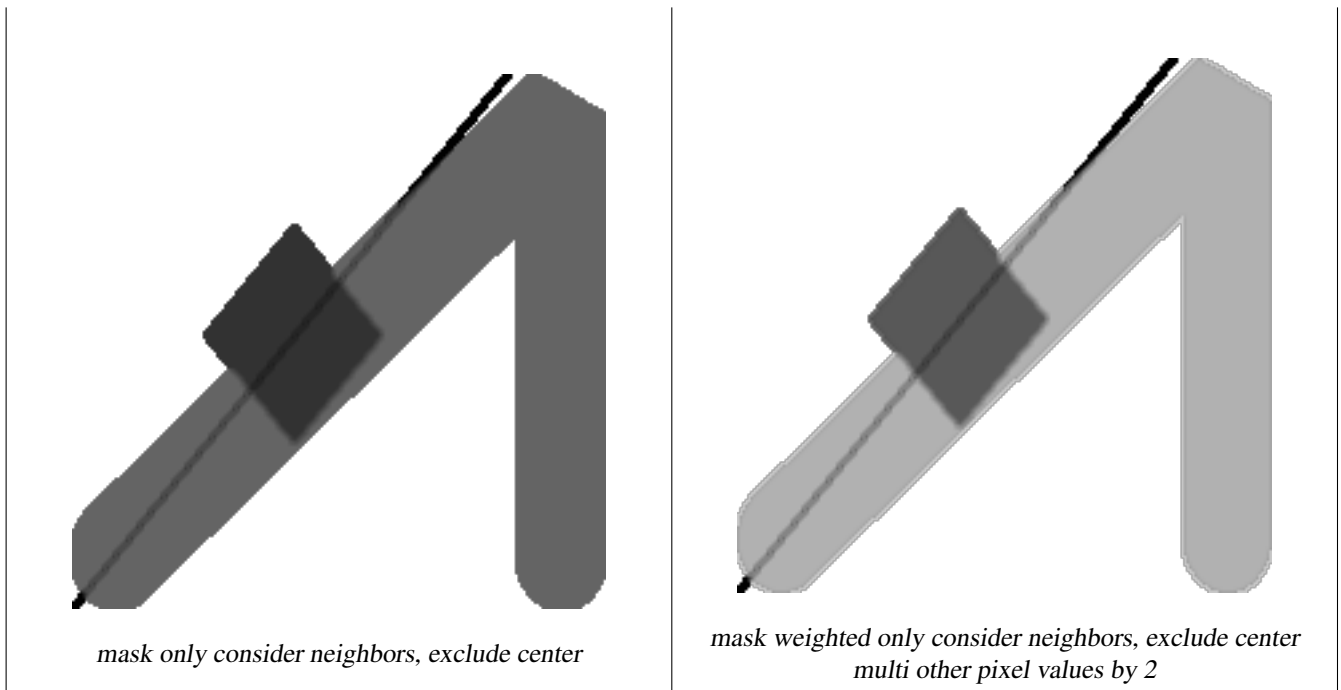
```

WITH foo AS (SELECT
 ST_SetBandNoDataValue(
ST_SetValue(ST_SetValue(ST_AsRaster(
 ST_Buffer(
 ST_GeomFromText('LINESTRING(50 50,100 90,100 50)'), 5,'join=bevel'),
 200,200,ARRAY['8BUI'], ARRAY[100], ARRAY[0]), ST_Buffer('POINT(70 70) ':: geometry,10,'quad_segs=1') ,50),
 'LINESTRING(20 20, 100 100, 150 98) '::geometry,1),0) AS rast)
SELECT 'original' AS title, rast
FROM foo
UNION ALL
SELECT 'no mask mean value' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[], ←
 int[], text[]) '::regprocedure) AS rast
FROM foo
UNION ALL
SELECT 'mask only consider neighbors, exclude center' AS title, ST_MapAlgebra(rast,1,' ←
 ST_mean4ma(double precision[], int[], text[]) '::regprocedure,
 '{{1,1,1}, {1,0,1}, {1,1,1}}' ::double precision[], false) As rast
FROM foo

UNION ALL
SELECT 'mask weighted only consider neighbors, exclude center multi otehr pixel values by ←
 2' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[], int[], text[]) ':: ←
 regprocedure,
 '{{2,2,2}, {2,0,2}, {2,2,2}}' ::double precision[], true) As rast
FROM foo;

```





### See Also

[rastbandarg](#), [ST\\_Union](#), [ST\\_MapAlgebra \(expression version\)](#)

## 10.12.6 ST\_MapAlgebra (expression version)

**ST\_MapAlgebra (expression version)** — Expression version - Returns a one-band raster given one or two input rasters, band indexes and one or more user-specified SQL expressions.

### Synopsis

```
raster ST_MapAlgebra(raster rast, integer nband, text pixeltype, text expression, double precision nodataval=NULL);
raster ST_MapAlgebra(raster rast, text pixeltype, text expression, double precision nodataval=NULL);
raster ST_MapAlgebra(raster rast1, integer nband1, raster rast2, integer nband2, text expression, text pixeltype=NULL, text
extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
raster ST_MapAlgebra(raster rast1, raster rast2, text expression, text pixeltype=NULL, text extenttype=INTERSECTION, text
nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
```

### Description

Expression version - Returns a one-band raster given one or two input rasters, band indexes and one or more user-specified SQL expressions.

Availability: 2.1.0

### Description: Variants 1 and 2 (one raster)

Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation defined by the `expression` on the input raster (`rast`). If `nband` is not provided, band 1 is assumed. The new raster will have the same georeference, width, and height as the original raster but will only have one band.

If `pixeltype` is passed in, then the new raster will have a band of that pixeltype. If `pixeltype` is passed NULL, then the new raster band will have the same pixeltype as the input `rast` band.

- Keywords permitted for `expression`

1. `[rast]` - Pixel value of the pixel of interest
2. `[rast.val]` - Pixel value of the pixel of interest
3. `[rast.x]` - 1-based pixel column of the pixel of interest
4. `[rast.y]` - 1-based pixel row of the pixel of interest

#### Description: Variants 3 and 4 (two raster)

Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation to the two bands defined by the `expression` on the two input raster bands `rast1`, (`rast2`). If no `band1`, `band2` is specified band 1 is assumed. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster. The resulting raster will have the extent defined by the `extenttype` parameter.

**expression** A PostgreSQL algebraic expression involving the two rasters and PostgreSQL defined functions/operators that will define the pixel value when pixels intersect. e.g. `(([rast1] + [rast2])/2.0)::integer`

**pixeltype** The resulting pixel type of the output raster. Must be one listed in [ST\\_BandPixelType](#), left out or set to NULL. If not passed in or set to NULL, will default to the pixeltype of the first raster.

**extenttype** Controls the extent of resulting raster

1. `INTERSECTION` - The extent of the new raster is the intersection of the two rasters. This is the default.
2. `UNION` - The extent of the new raster is the union of the two rasters.
3. `FIRST` - The extent of the new raster is the same as the one of the first raster.
4. `SECOND` - The extent of the new raster is the same as the one of the second raster.

**nodata1expr** An algebraic expression involving only `rast2` or a constant that defines what to return when pixels of `rast1` are nodata values and spatially corresponding `rast2` pixels have values.

**nodata2expr** An algebraic expression involving only `rast1` or a constant that defines what to return when pixels of `rast2` are nodata values and spatially corresponding `rast1` pixels have values.

**nodatanodataval** A numeric constant to return when spatially corresponding `rast1` and `rast2` pixels are both nodata values.

- Keywords permitted in `expression`, `nodata1expr` and `nodata2expr`

1. `[rast1]` - Pixel value of the pixel of interest from `rast1`
2. `[rast1.val]` - Pixel value of the pixel of interest from `rast1`
3. `[rast1.x]` - 1-based pixel column of the pixel of interest from `rast1`
4. `[rast1.y]` - 1-based pixel row of the pixel of interest from `rast1`
5. `[rast2]` - Pixel value of the pixel of interest from `rast2`
6. `[rast2.val]` - Pixel value of the pixel of interest from `rast2`
7. `[rast2.x]` - 1-based pixel column of the pixel of interest from `rast2`
8. `[rast2.y]` - 1-based pixel row of the pixel of interest from `rast2`

#### Examples: Variants 1 and 2

```
WITH foo AS (
 SELECT ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, 1, 0, 0, 0), '32BF'::text, 1, -1) ←
 AS rast
)
SELECT
 ST_MapAlgebra(rast, 1, NULL, 'ceil([rast]*[rast.x]/[rast.y]+[rast.val])')
FROM foo;
```

**Examples: Variant 3 and 4**

```

WITH foo AS (
 SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ←
 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI'::text, 100, 0) AS rast ←
 UNION ALL
 SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, 1, -1, ←
 0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI'::text, 300, 0) AS rast
)
SELECT
 ST_MapAlgebra(
 t1.rast, 2,
 t2.rast, 1,
 '([rast2] + [rast1.val]) / 2'
) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
 AND t2.rid = 2;

```

**See Also**

[rastbandarg](#), [ST\\_Union](#), [ST\\_MapAlgebra \(callback function version\)](#)

**10.12.7 ST\_MapAlgebraExpr**

**ST\_MapAlgebraExpr** — 1 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the input raster band and of pixeltype provided. Band 1 is assumed if no band is specified.

**Synopsis**

raster **ST\_MapAlgebraExpr**(raster rast, integer band, text pixeltype, text expression, double precision nodataval=NULL);  
raster **ST\_MapAlgebraExpr**(raster rast, text pixeltype, text expression, double precision nodataval=NULL);

**Description****Warning**

**ST\_MapAlgebraExpr** is deprecated as of 2.1.0. Use [ST\\_MapAlgebra \(expression version\)](#) instead.

Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation defined by the `expression` on the input raster (`rast`). If no band is specified band 1 is assumed. The new raster will have the same georeference, width, and height as the original raster but will only have one band.

If `pixeltype` is passed in, then the new raster will have a band of that pixeltype. If `pixeltype` is passed NULL, then the new raster band will have the same pixeltype as the input `rast` band.

In the expression you can use the term `[rast]` to refer to the pixel value of the original band, `[rast.x]` to refer to the 1-based pixel column index, `[rast.y]` to refer to the 1-based pixel row index.

Availability: 2.0.0

## Examples

Create a new 1 band raster from our original that is a function of modulo 2 of the original raster band.

```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
UPDATE dummy_rast SET map_rast = ST_MapAlgebraExpr(rast,NULL,'mod([rast]::numeric,2)') ←
 WHERE rid = 2;

SELECT
 ST_Value(rast,1,i,j) As origval,
 ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 3) AS i
CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

| origval | mapval |
|---------|--------|
| 253     | 1      |
| 254     | 0      |
| 253     | 1      |
| 253     | 1      |
| 254     | 0      |
| 254     | 0      |
| 250     | 0      |
| 254     | 0      |
| 254     | 0      |

Create a new 1 band raster of pixel-type 2BUI from our original that is reclassified and set the nodata value to be 0.

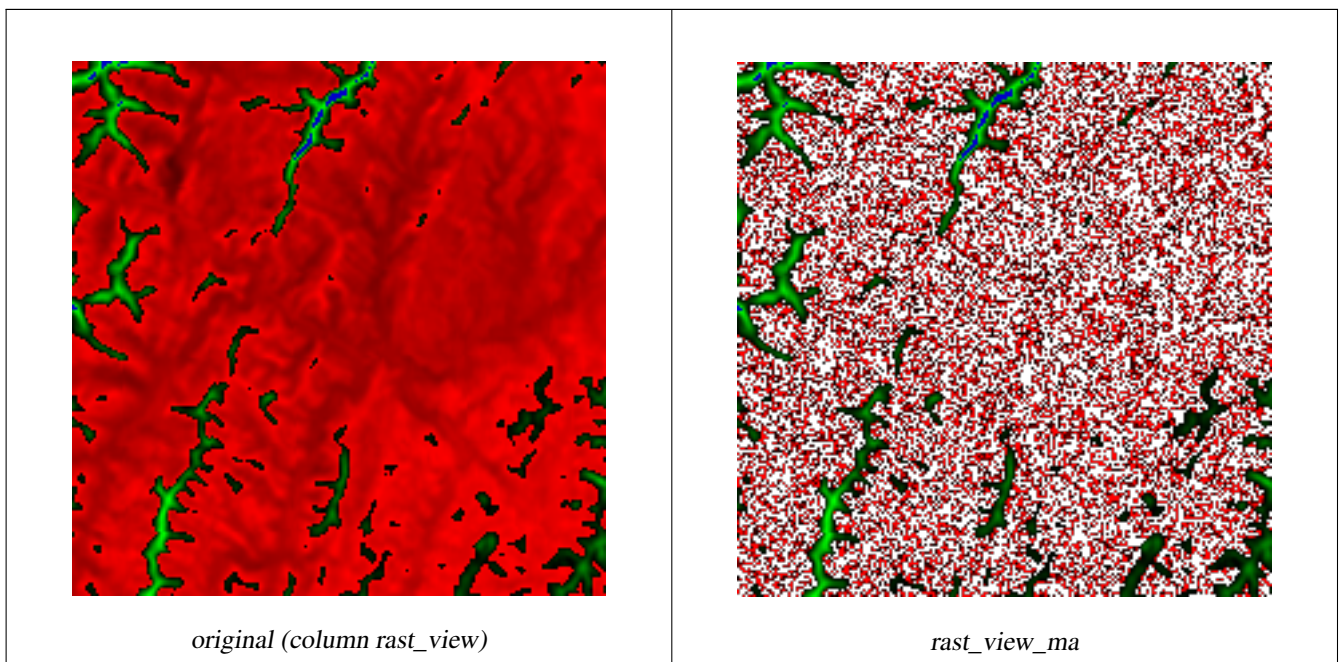
```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
UPDATE dummy_rast SET
 map_rast2 = ST_MapAlgebraExpr(rast,'2BUI'::text,'CASE WHEN [rast] BETWEEN 100 and 250 ←
 THEN 1 WHEN [rast] = 252 THEN 2 WHEN [rast] BETWEEN 253 and 254 THEN 3 ELSE 0 END':: ←
 text, '0')
WHERE rid = 2;
```

```
SELECT DISTINCT
 ST_Value(rast,1,i,j) As origval,
 ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 5) AS i
CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;
```

| origval | mapval |
|---------|--------|
| 249     | 1      |
| 250     | 1      |
| 251     |        |
| 252     | 2      |
| 253     | 3      |
| 254     | 3      |

```
SELECT
 ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast
WHERE rid = 2;
```

| b1pixtyp |
|----------|
| 2BUI     |



Create a new 3 band raster same pixel type from our original 3 band raster with first band altered by map algebra and remaining 2 bands unaltered.

```
SELECT
 ST_AddBand(
 ST_AddBand(
 ST_AddBand(
 ST_MakeEmptyRaster(rast_view),
 ST_MapAlgebraExpr(rast_view,1,NULL,'tan([rast])*[rast]')
),
 ST_Band(rast_view,2)
),
 ST_Band(rast_view, 3)
) As rast_view_ma
FROM wind
WHERE rid=167;
```

#### See Also

[ST\\_MapAlgebraExpr](#), [ST\\_MapAlgebraFct](#), [ST\\_BandPixelType](#), [ST\\_GeoReference](#), [ST\\_Value](#)

### 10.12.8 ST\_MapAlgebraExpr

**ST\_MapAlgebraExpr** — 2 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the two input raster bands and of pixeltype provided. band 1 of each raster is assumed if no band numbers are specified. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster and have its extent defined by the "extenttype" parameter. Values for "extenttype" can be: INTERSECTION, UNION, FIRST, SECOND.

#### Synopsis

raster **ST\_MapAlgebraExpr**(raster rast1, raster rast2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

raster **ST\_MapAlgebraExpr**(raster rast1, integer band1, raster rast2, integer band2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

## Description



### Warning

**ST\_MapAlgebraExpr** is deprecated as of 2.1.0. Use **ST\_MapAlgebra (expression version)** instead.

Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation to the two bands defined by the expression on the two input raster bands `rast1`, (`rast2`). If no `band1`, `band2` is specified band 1 is assumed. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster. The resulting raster will have the extent defined by the `extenttype` parameter.

**expression** A PostgreSQL algebraic expression involving the two rasters and PostgreSQL defined functions/operators that will define the pixel value when pixels intersect. e.g. `(([rast1] + [rast2])/2.0)::integer`

**pixeltype** The resulting pixel type of the output raster. Must be one listed in **ST\_BandPixelType**, left out or set to NULL. If not passed in or set to NULL, will default to the pixeltype of the first raster.

**extenttype** Controls the extent of resulting raster

1. **INTERSECTION** - The extent of the new raster is the intersection of the two rasters. This is the default.
2. **UNION** - The extent of the new raster is the union of the two rasters.
3. **FIRST** - The extent of the new raster is the same as the one of the first raster.
4. **SECOND** - The extent of the new raster is the same as the one of the second raster.

**nodata1expr** An algebraic expression involving only `rast2` or a constant that defines what to return when pixels of `rast1` are nodata values and spatially corresponding `rast2` pixels have values.

**nodata2expr** An algebraic expression involving only `rast1` or a constant that defines what to return when pixels of `rast2` are nodata values and spatially corresponding `rast1` pixels have values.

**nodatanodataval** A numeric constant to return when spatially corresponding `rast1` and `rast2` pixels are both nodata values.

If `pixeltype` is passed in, then the new raster will have a band of that pixeltype. If `pixeltype` is passed NULL or no pixel type specified, then the new raster band will have the same pixeltype as the input `rast1` band.

Use the term `[rast1.val]` `[rast2.val]` to refer to the pixel value of the original raster bands and `[rast1.x]`, `[rast1.y]` etc. to refer to the column / row positions of the pixels.

Availability: 2.0.0

## Example: 2 Band Intersection and Union

Create a new 1 band raster from our original that is a function of modulo 2 of the original raster band.

```
--Create a cool set of rasters --
DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

-- Insert some cool shapes around Boston in Massachusetts state plane meters --
INSERT INTO fun_shapes(fun_name, rast)
VALUES ('ref', ST_AsRaster(ST_MakeEnvelope(235229, 899970, 237229, 901930,26986),200,200,'8 ←
 BUI',0,0));

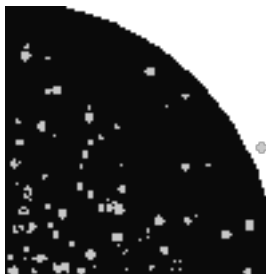
INSERT INTO fun_shapes(fun_name,rast)
WITH ref(rast) AS (SELECT rast FROM fun_shapes WHERE fun_name = 'ref')
SELECT 'area' AS fun_name, ST_AsRaster(ST_Buffer(ST_SetSRID(ST_Point(236229, 900930),26986) ←
 , 1000),
```

```

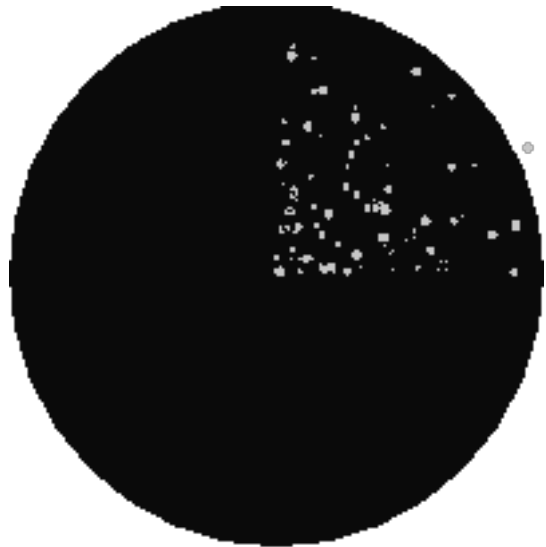
 ref.rast,'8BUI', 10, 0) As rast
FROM ref
UNION ALL
SELECT 'rand bubbles',
 ST_AsRaster(
 (SELECT ST_Collect(geom)
 FROM (SELECT ST_Buffer(ST_SetSRID(ST_Point(236229 + i*random()*100, 900930 + j*random()*100),26986), random()*20) As geom
 FROM generate_series(1,10) As i, generate_series(1,10) As j
) As foo), ref.rast,'8BUI', 200, 0)
FROM ref;

--map them -
SELECT ST_MapAlgebraExpr(
 area.rast, bub.rast, '[rast2.val]', '8BUI', 'INTERSECTION', '[rast2.val]', '[rast1.val]') As interrast,
 ST_MapAlgebraExpr(
 area.rast, bub.rast, '[rast2.val]', '8BUI', 'UNION', '[rast2.val]', '[rast1.val]') As unionrast
FROM
 (SELECT rast FROM fun_shapes WHERE
 fun_name = 'area') As area
CROSS JOIN (SELECT rast
FROM fun_shapes WHERE
fun_name = 'rand bubbles') As bub

```



*mapalgebra intersection*



*map algebra union*

### Example: Overlaying rasters on a canvas as separate bands

```

-- we use ST_AsPNG to render the image so all single band ones look grey --
WITH mygeoms
 AS (SELECT 2 As bnum, ST_Buffer(ST_Point(1,5),10) As geom
 UNION ALL
 SELECT 3 AS bnum,
 ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10,'join=bevel') As geom
 UNION ALL

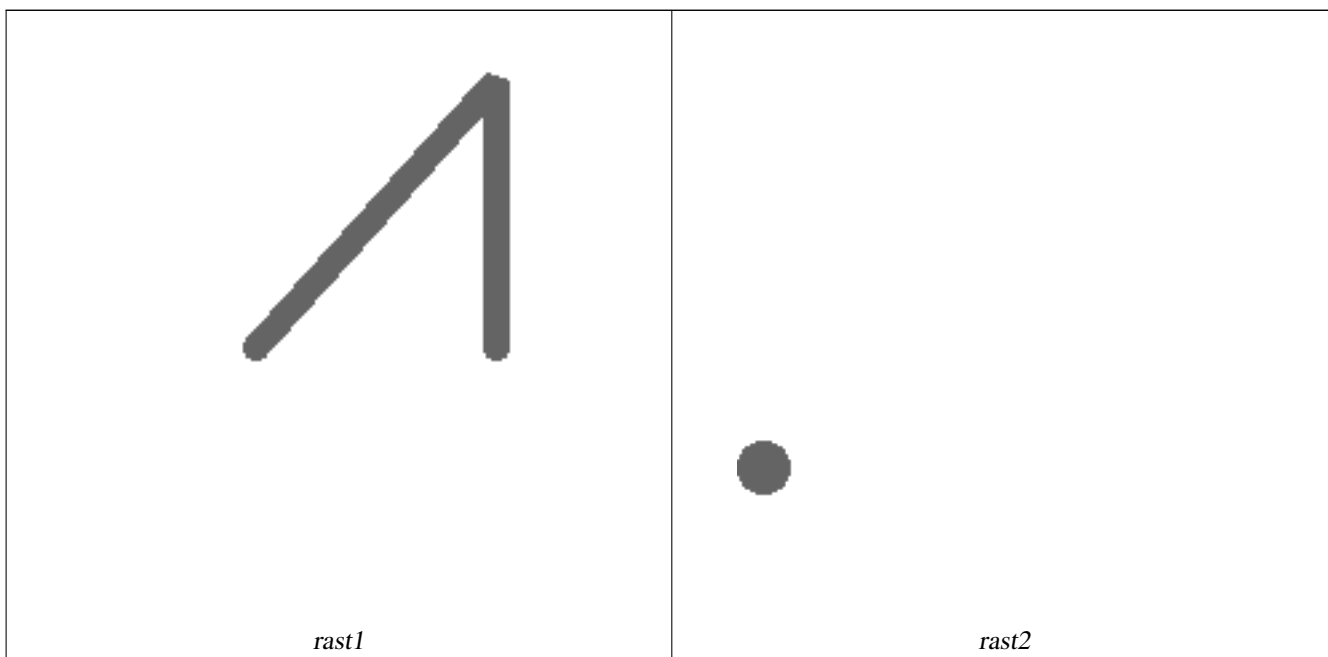
```

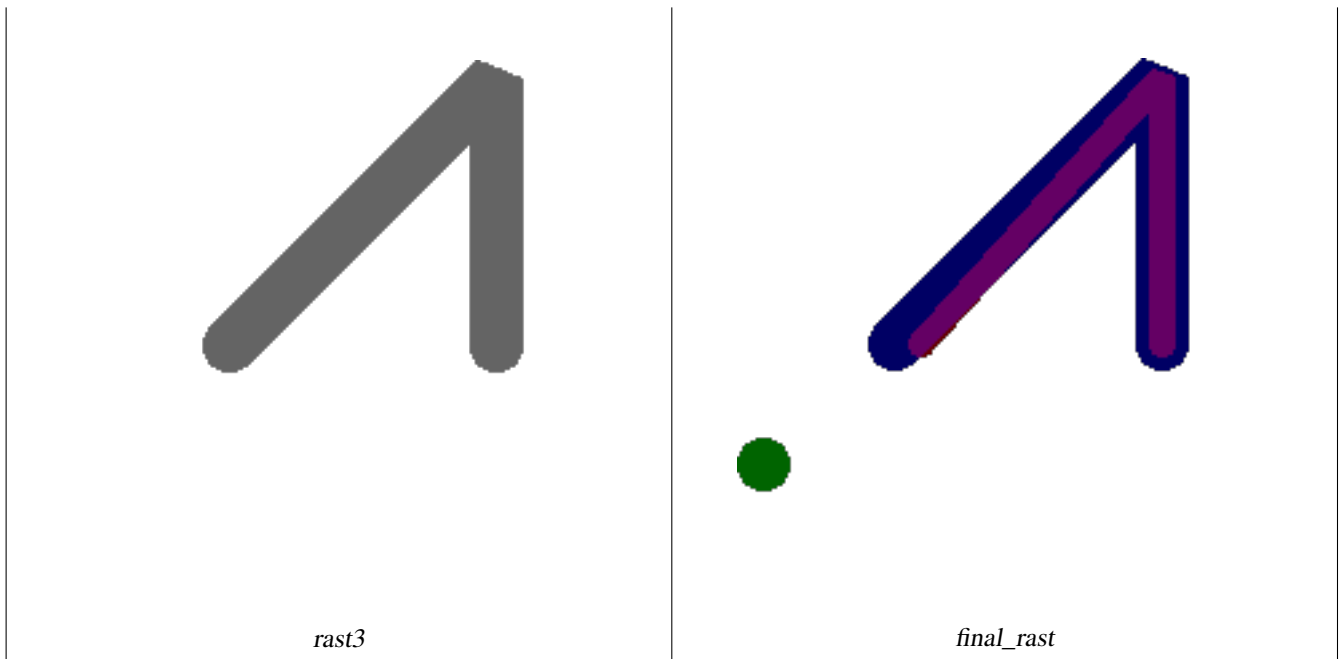


```

SELECT 1 As bnum,
 ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'), 5,'join= ↵
 bevel') As geom
),
-- define our canvas to be 1 to 1 pixel to geometry
canvas
AS (SELECT ST_AddBand(ST_MakeEmptyRaster(200,
 200,
 ST_XMin(e)::integer, ST_YMax(e)::integer, 1, -1, 0, 0) , '8BUI'::text,0) As rast
FROM (SELECT ST_Extent(geom) As e,
 Max(ST_SRID(geom)) As srid
 from mygeoms
) As foo
),
rbands AS (SELECT ARRAY(SELECT ST_MapAlgebraExpr(canvas.rast, ST_AsRaster(m.geom, canvas ↵
.rast, '8BUI', 100),
 '[rast2.val]', '8BUI', 'FIRST', '[rast2.val]', '[rast1.val]') As rast
 FROM mygeoms AS m CROSS JOIN canvas
 ORDER BY m.bnum) As rasts
)
SELECT rasts[1] As rast1 , rasts[2] As rast2, rasts[3] As rast3, ST_AddBand(
 ST_AddBand(rasts[1],rasts[2]), rasts[3]) As final_rast
FROM rbands;

```





### Example: Overlay 2 meter boundary of select parcels over an aerial imagery

```
-- Create new 3 band raster composed of first 2 clipped bands, and overlay of 3rd band with ←
 our geometry
-- This query took 3.6 seconds on PostGIS windows 64-bit install
WITH pr AS
-- Note the order of operation: we clip all the rasters to dimensions of our region
(SELECT ST_Clip(rast,ST_Expand(geom,50)) As rast, g.geom
 FROM aerals.o_2_boston AS r INNER JOIN
-- union our parcels of interest so they form a single geometry we can later intersect with
 (SELECT ST_Union(ST_Transform(geom,26986)) AS geom
 FROM landparcels WHERE pid IN('0303890000', '0303900000')) As g
 ON ST_Intersects(rast::geometry, ST_Expand(g.geom,50))
),
-- we then union the raster shards together
-- ST_Union on raster is kinda of slow but much faster the smaller you can get the rasters
-- therefore we want to clip first and then union
prunion AS
(SELECT ST_AddBand(NULL, ARRAY[ST_Union(rast,1),ST_Union(rast,2),ST_Union(rast,3)]) As ←
 clipped,geom
 FROM pr
 GROUP BY geom)
-- return our final raster which is the unioned shard with
-- with the overlay of our parcel boundaries
-- add first 2 bands, then mapalgebra of 3rd band + geometry
SELECT ST_AddBand(ST_Band(clipped,ARRAY[1,2])
 , ST_MapAlgebraExpr(ST_Band(clipped,3), ST_AsRaster(ST_Buffer(ST_Boundary(geom),2), ←
 clipped, '8BUI',250),
 '[rast2.val]', '8BUI', 'FIRST', '[rast2.val]', '[rast1.val]')) As rast
 FROM prunion;
```



*The blue lines are the boundaries of select parcels*

#### See Also

[ST\\_MapAlgebraExpr](#), [ST\\_AddBand](#), [ST\\_AsPNG](#), [ST\\_AsRaster](#), [ST\\_MapAlgebraFct](#), [ST\\_BandPixelType](#), [ST\\_GeoReference](#), [ST\\_Value](#), [ST\\_Union](#), [ST\\_Union](#)

### 10.12.9 ST\_MapAlgebraFct

**ST\_MapAlgebraFct** — 1 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the input raster band and of pixeltype provided. Band 1 is assumed if no band is specified.

#### Synopsis

```
raster ST_MapAlgebraFct(raster rast, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster ST_MapAlgebraFct(raster rast, text pixeltype, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster ST_MapAlgebraFct(raster rast, integer band, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, integer band, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster ST_MapAlgebraFct(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);
```

#### Description



#### Warning

**ST\_MapAlgebraFct** is deprecated as of 2.1.0. Use **ST\_MapAlgebra** (callback function version) instead.

Creates a new one band raster formed by applying a valid PostgreSQL function specified by the `onerasteruserfunc` on the input raster (`rast`). If no band is specified, band 1 is assumed. The new raster will have the same georeference, width, and height as the original raster but will only have one band.

If `pixeltype` is passed in, then the new raster will have a band of that pixeltype. If `pixeltype` is passed NULL, then the new raster band will have the same pixeltype as the input `rast` band.

The `onerasteruserfunc` parameter must be the name and signature of a SQL or PL/pgSQL function, cast to a regprocedure. A very simple and quite useless PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION simple_function(pixel FLOAT, pos INTEGER[], VARIADIC args TEXT ↔
[])
RETURNS FLOAT
AS $$ BEGIN
 RETURN 0.0;
END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

The `userfunction` may accept two or three arguments: a float value, an optional integer array, and a variadic text array. The first argument is the value of an individual raster cell (regardless of the raster datatype). The second argument is the position of the current processing cell in the form '`{x,y}`'. The third argument indicates that all remaining parameters to `ST_MapAlgebraFct` shall be passed through to the `userfunction`.

Passing a regprocedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'simple_function(float,integer[],text[]) '::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

The third argument to the `userfunction` is a variadic text array. All trailing text arguments to any `ST_MapAlgebraFct` call are passed through to the specified `userfunction`, and are contained in the `args` argument.



#### Note

For more information about the VARIADIC keyword, please refer to the PostgreSQL documentation and the "SQL Functions with Variable Numbers of Arguments" section of [Query Language \(SQL\) Functions](#).



#### Note

The `text[]` argument to the `userfunction` is required, regardless of whether you choose to pass any arguments to your user function for processing or not.

Availability: 2.0.0

## Examples

Create a new 1 band raster from our original that is a function of modulo 2 of the original raster band.

```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
CREATE FUNCTION mod_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS $$
BEGIN
 RETURN pixel::integer % 2;
END;
$$
```

```
LANGUAGE 'plpgsql' IMMUTABLE;
```

```
UPDATE dummy_rast SET map_rast = ST_MapAlgebraFct(rast,NULL,'mod_fct(float,integer[],text ←
[])'::regprocedure) WHERE rid = 2;
```

```
SELECT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

| origval | mapval |
|---------|--------|
| 253     | 1      |
| 254     | 0      |
| 253     | 1      |
| 253     | 1      |
| 254     | 0      |
| 254     | 0      |
| 250     | 0      |
| 254     | 0      |
| 254     | 0      |

Create a new 1 band raster of pixel-type 2BUI from our original that is reclassified and set the nodata value to a passed parameter to the user function (0).

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
```

```
CREATE FUNCTION classify_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
```

```
AS
```

```
$$
```

```
DECLARE
```

```
 nodata float := 0;
```

```
BEGIN
```

```
 IF NOT args[1] IS NULL THEN
```

```
 nodata := args[1];
```

```
 END IF;
```

```
 IF pixel < 251 THEN
```

```
 RETURN 1;
```

```
 ELSIF pixel = 252 THEN
```

```
 RETURN 2;
```

```
 ELSIF pixel > 252 THEN
```

```
 RETURN 3;
```

```
 ELSE
```

```
 RETURN nodata;
```

```
 END IF;
```

```
END;
```

```
$$
```

```
LANGUAGE 'plpgsql';
```

```
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ←
[],text[])'::regprocedure, '0') WHERE rid = 2;
```

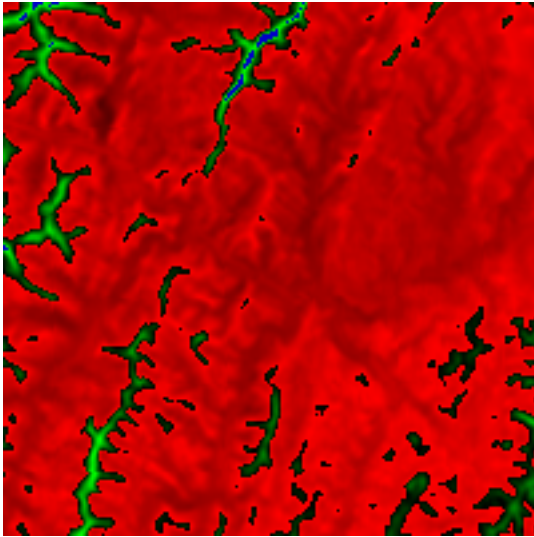
```
SELECT DISTINCT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 5) AS i CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;
```

| origval | mapval |
|---------|--------|
| 249     | 1      |
| 250     | 1      |
| 251     |        |
| 252     | 2      |
| 253     | 3      |
| 254     | 3      |

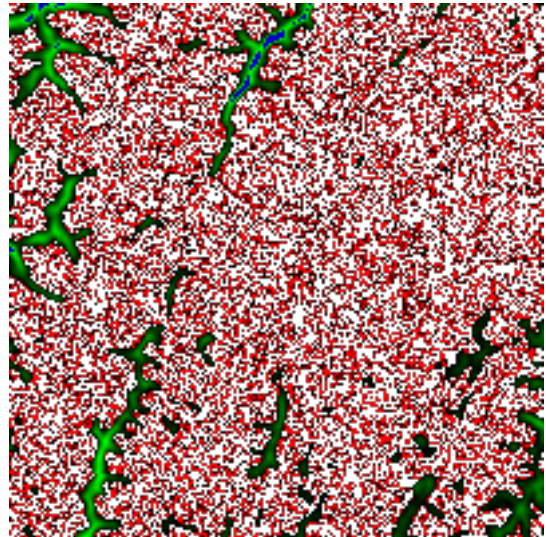
```
SELECT ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast WHERE rid = 2;
```

```
b1pixtyp

2BUI
```



*original (column rast-view)*



*rast\_view\_ma*

Create a new 3 band raster same pixel type from our original 3 band raster with first band altered by map algebra and remaining 2 bands unaltered.

```
CREATE FUNCTION rast_plus_tan(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
BEGIN
 RETURN tan(pixel) * pixel;
END;
$$
LANGUAGE 'plpgsql';

SELECT ST_AddBand(
 ST_AddBand(
 ST_AddBand(
 ST_MakeEmptyRaster(rast_view),
 ST_MapAlgebraFct(rast_view,1,NULL,'rast_plus_tan(float,integer[],text[])':: ↵
 regprocedure)
),
 ST_Band(rast_view,2)
),
 ST_Band(rast_view, 3) As rast_view_ma
)
FROM wind
WHERE rid=167;
```

**See Also**

[ST\\_MapAlgebraExpr](#), [ST\\_BandPixelType](#), [ST\\_GeoReference](#), [ST\\_SetValue](#)

**10.12.10 ST\_MapAlgebraFct**

**ST\_MapAlgebraFct** — 2 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the 2 input raster bands and of pixeltype provided. Band 1 is assumed if no band is specified. Extent type defaults to INTERSECTION if not specified.

**Synopsis**

raster **ST\_MapAlgebraFct**(raster rast1, raster rast2, regprocedure tworastuserfunc, text pixeltype=same\_as\_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);

raster **ST\_MapAlgebraFct**(raster rast1, integer band1, raster rast2, integer band2, regprocedure tworastuserfunc, text pixeltype=same\_as\_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);

**Description****Warning**

**ST\_MapAlgebraFct** is deprecated as of 2.1.0. Use [ST\\_MapAlgebra \(callback function version\)](#) instead.

Creates a new one band raster formed by applying a valid PostgreSQL function specified by the `tworastuserfunc` on the input raster `rast1`, `rast2`. If no `band1` or `band2` is specified, band 1 is assumed. The new raster will have the same georeference, width, and height as the original rasters but will only have one band.

If `pixeltype` is passed in, then the new raster will have a band of that pixeltype. If `pixeltype` is passed NULL or left out, then the new raster band will have the same pixeltype as the input `rast1` band.

The `tworastuserfunc` parameter must be the name and signature of an SQL or PL/pgSQL function, cast to a regprocedure. An example PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION simple_function_for_two_rasters(pixel1 FLOAT, pixel2 FLOAT, pos ←
 INTEGER[], VARIADIC args TEXT[])
 RETURNS FLOAT
 AS $$ BEGIN
 RETURN 0.0;
 END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

The `tworastuserfunc` may accept three or four arguments: a double precision value, a double precision value, an optional integer array, and a variadic text array. The first argument is the value of an individual raster cell in `rast1` (regardless of the raster datatype). The second argument is an individual raster cell value in `rast2`. The third argument is the position of the current processing cell in the form '`{x,y}`'. The fourth argument indicates that all remaining parameters to **ST\_MapAlgebraFct** shall be passed through to the `tworastuserfunc`.

Passing a regprocedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'simple_function(double precision, double precision, integer[], text[])':regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

The fourth argument to the `tworastuserfunc` is a variadic text array. All trailing text arguments to any **ST\_MapAlgebraFct** call are passed through to the specified `tworastuserfunc`, and are contained in the `userargs` argument.

**Note**

For more information about the VARIADIC keyword, please refer to the PostgreSQL documentation and the "SQL Functions with Variable Numbers of Arguments" section of [Query Language \(SQL\) Functions](#).

**Note**

The text[] argument to the `tworastuserfunc` is required, regardless of whether you choose to pass any arguments to your user function for processing or not.

Availability: 2.0.0

**Example: Overlaying rasters on a canvas as separate bands**

```
-- define our user defined function --
CREATE OR REPLACE FUNCTION raster_mapalgebra_union(
 rast1 double precision,
 rast2 double precision,
 pos integer[],
 VARIADIC userargs text[]
)
RETURNS double precision
AS $$
DECLARE
BEGIN
 CASE
 WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
 RETURN ((rast1 + rast2)/2.);
 WHEN rast1 IS NULL AND rast2 IS NULL THEN
 RETURN NULL;
 WHEN rast1 IS NULL THEN
 RETURN rast2;
 ELSE
 RETURN rast1;
 END CASE;

 RETURN NULL;
END;
$$ LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- prep our test table of rasters
DROP TABLE IF EXISTS map_shapes;
CREATE TABLE map_shapes(rid serial PRIMARY KEY, rast raster, bnum integer, descrip text);
INSERT INTO map_shapes(rast,bnum, descrip)
WITH mygeoms
 AS (SELECT 2 As bnum, ST_Buffer(ST_Point(90,90),30) As geom, 'circle' As descrip
 UNION ALL
 SELECT 3 AS bnum,
 ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 15) As geom, ←
 'big road' As descrip
 UNION ALL
 SELECT 1 As bnum,
 ST_Translate(ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'), ←
 8,'join=bevel'), 10,-6) As geom, 'small road' As descrip
),
-- define our canvas to be 1 to 1 pixel to geometry
canvas
 AS (SELECT ST_AddBand(ST_MakeEmptyRaster(250,
```



```

 250,
 ST_XMin(e)::integer, ST_YMax(e)::integer, 1, -1, 0, 0) , '8BUI'::text,0) As rast
 FROM (SELECT ST_Extent(geom) As e,
 Max(ST_SRID(geom)) As srid
 from mygeoms
) As foo
)
-- return our rasters aligned with our canvas
SELECT ST_AsRaster(m.geom, canvas.rast, '8BUI', 240) As rast, bnum, descrip
 FROM mygeoms AS m CROSS JOIN canvas
UNION ALL
SELECT canvas.rast, 4, 'canvas'
FROM canvas;

-- Map algebra on single band rasters and then collect with ST_AddBand
INSERT INTO map_shapes(rast,bnum,descrip)
SELECT ST_AddBand(ST_AddBand(rasts[1], rasts[2]),rasts[3]), 4, 'map bands overlay fct union ←
 (canvas)'
 FROM (SELECT ARRAY(SELECT ST_MapAlgebraFct(m1.rast, m2.rast,
 'raster_mapalgebra_union(double precision, double precision, integer[], text[]) ←
 '::regprocedure, '8BUI', 'FIRST')
 FROM map_shapes As m1 CROSS JOIN map_shapes As m2
 WHERE m1.descrip = 'canvas' AND m2.descrip <> 'canvas' ORDER BY m2.bnum) As rasts) As ←
 foo;

```



*map bands overlay (canvas) (R: small road, G: circle, B: big road)*

### User Defined function that takes extra args

```

CREATE OR REPLACE FUNCTION raster_mapalgebra_userargs(
 rast1 double precision,
 rast2 double precision,
 pos integer[],
 VARIADIC userargs text[]

```

```

)
RETURNS double precision
AS $$
DECLARE
BEGIN
 CASE
 WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
 RETURN least(userargs[1]::integer, (rast1 + rast2)/2.);
 WHEN rast1 IS NULL AND rast2 IS NULL THEN
 RETURN userargs[2]::integer;
 WHEN rast1 IS NULL THEN
 RETURN greatest(rast2, random()*userargs[3]::integer)::integer;
 ELSE
 RETURN greatest(rast1, random()*userargs[4]::integer)::integer;
 END CASE;

 RETURN NULL;
END;
$$ LANGUAGE 'plpgsql' VOLATILE COST 1000;

SELECT ST_MapAlgebraFct(m1.rast, 1, m1.rast, 3,
 'raster_mapalgebra_userargs(double precision, double precision, integer[], text ←
 [])'::regprocedure,
 '8BUI', 'INTERSECT', '100','200','200','0')
 FROM map_shapes As m1
WHERE m1.descrip = 'map bands overlay fct union (canvas)';

```



*user defined with extra args and different bands from same raster*

#### See Also

[ST\\_MapAlgebraExpr](#), [ST\\_BandPixelType](#), [ST\\_GeoReference](#), [ST\\_SetValue](#)

### 10.12.11 ST\_MapAlgebraFctNgb

**ST\_MapAlgebraFctNgb** — 1-band version: Map Algebra Nearest Neighbor using user-defined PostgreSQL function. Return a raster which values are the result of a PLPGSQL user function involving a neighborhood of values from the input raster band.

#### Synopsis

raster **ST\_MapAlgebraFctNgb**(raster rast, integer band, text pixeltype, integer ngbwidth, integer ngbheight, regprocedure onerastngbuserfunc, text nodatamode, text[] VARIADIC args);

#### Description



#### Warning

**ST\_MapAlgebraFctNgb** is deprecated as of 2.1.0. Use **ST\_MapAlgebra (callback function version)** instead.

(one raster version) Return a raster which values are the result of a PLPGSQL user function involving a neighborhood of values from the input raster band. The user function takes the neighborhood of pixel values as an array of numbers, for each pixel, returns the result from the user function, replacing pixel value of currently inspected pixel with the function result.

**rast** Raster on which the user function is evaluated.

**band** Band number of the raster to be evaluated. Default to 1.

**pixeltype** The resulting pixel type of the output raster. Must be one listed in **ST\_BandPixelType** or left out or set to NULL. If not passed in or set to NULL, will default to the pixeltype of the `rast`. Results are truncated if they are larger than what is allowed for the pixeltype.

**ngbwidth** The width of the neighborhood, in cells.

**ngbheight** The height of the neighborhood, in cells.

**onerastngbuserfunc** PLPGSQL/psql user function to apply to neighborhood pixels of a single band of a raster. The first element is a 2-dimensional array of numbers representing the rectangular pixel neighborhood

**nodatamode** Defines what value to pass to the function for a neighborhood pixel that is nodata or NULL

'ignore': any NODATA values encountered in the neighborhood are ignored by the computation -- this flag must be sent to the user callback function, and the user function decides how to ignore it.

'NULL': any NODATA values encountered in the neighborhood will cause the resulting pixel to be NULL -- the user callback function is skipped in this case.

'value': any NODATA values encountered in the neighborhood are replaced by the reference pixel (the one in the center of the neighborhood). Note that if this value is NODATA, the behavior is the same as 'NULL' (for the affected neighborhood)

**args** Arguments to pass into the user function.

Availability: 2.0.0

#### Examples

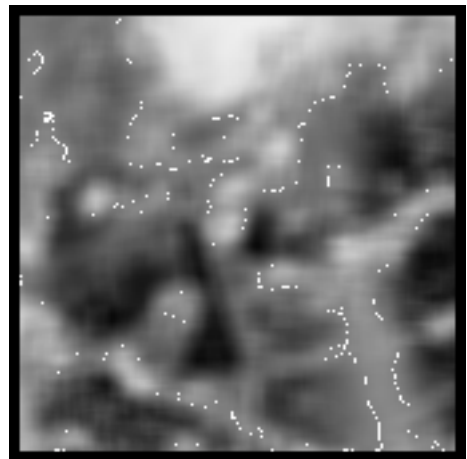
Examples utilize the katrina raster loaded as a single tile described in [http://trac.osgeo.org/gdal/wiki/frmts\\_wtkraster.html](http://trac.osgeo.org/gdal/wiki/frmts_wtkraster.html) and then prepared in the **ST\_Rescale** examples

```
--
-- A simple 'callback' user function that averages up all the values in a neighborhood.
--
CREATE OR REPLACE FUNCTION rast_avg(matrix float[][], nodatamode text, variadic args text ←
[])
RETURNS float AS
$$
DECLARE
 _matrix float[][];
 x1 integer;
 x2 integer;
 y1 integer;
 y2 integer;
 sum float;
BEGIN
 _matrix := matrix;
 sum := 0;
 FOR x in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
 FOR y in array_lower(matrix, 2)..array_upper(matrix, 2) LOOP
 sum := sum + _matrix[x][y];
 END LOOP;
 END LOOP;
 RETURN (sum*1.0/(array_upper(matrix,1)*array_upper(matrix,2)))::integer ;
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- now we apply to our raster averaging pixels within 2 pixels of each other in X and Y ←
direction --
SELECT ST_MapAlgebraFctNgb(rast, 1, '8BUI', 4,4,
 'rast_avg(float[][], text, text[])':regprocedure, 'NULL', NULL) As nn_with_border
FROM katrinas_rescaled
limit 1;
```



*First band of our raster*



*new raster after averaging pixels withing 4x4 pixels of each other*

### See Also

[ST\\_MapAlgebraFct](#), [ST\\_MapAlgebraExpr](#), [ST\\_Rescale](#)

### 10.12.12 ST\_Reclass

**ST\_Reclass** — Creates a new raster composed of band types reclassified from original. The nband is the band to be changed. If nband is not specified assumed to be 1. All other bands are returned unchanged. Use case: convert a 16BUI band to a 8BUI and so forth for simpler rendering as viewable formats.

#### Synopsis

```
raster ST_Reclass(raster rast, integer nband, text reclassexpr, text pixeltype, double precision nodataval=NULL);
raster ST_Reclass(raster rast, reclassarg[] VARIADIC reclassargset);
raster ST_Reclass(raster rast, text reclassexpr, text pixeltype);
```

#### Description

Creates a new raster formed by applying a valid PostgreSQL algebraic operation defined by the `reclassexpr` on the input raster (`rast`). If no band is specified band 1 is assumed. The new raster will have the same georeference, width, and height as the original raster. Bands not designated will come back unchanged. Refer to [reclassarg](#) for description of valid reclassification expressions.

The bands of the new raster will have pixel type of `pixeltype`. If `reclassargset` is passed in then each `reclassarg` defines behavior of each band generated.

Availability: 2.0.0

#### Examples Basic

Create a new raster from the original where band 2 is converted from 8BUI to 4BUI and all values from 101-254 are set to nodata value.

```
ALTER TABLE dummy_rast ADD COLUMN reclass_rast raster;
UPDATE dummy_rast SET reclass_rast = ST_Reclass(rast,2,'0-87:1-10, 88-100:11-15, ←
 101-254:0-0', '4BUI',0) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast,2,i,j) As origval,
 ST_Value(reclass_rast, 2, i, j) As reclassval,
 ST_Value(reclass_rast, 2, i, j, false) As reclassval_include_nodata
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

| col | row | origval | reclassval | reclassval_include_nodata |
|-----|-----|---------|------------|---------------------------|
| 1   | 1   | 78      | 9          | 9                         |
| 2   | 1   | 98      | 14         | 14                        |
| 3   | 1   | 122     |            | 0                         |
| 1   | 2   | 96      | 14         | 14                        |
| 2   | 2   | 118     |            | 0                         |
| 3   | 2   | 180     |            | 0                         |
| 1   | 3   | 99      | 15         | 15                        |
| 2   | 3   | 112     |            | 0                         |
| 3   | 3   | 169     |            | 0                         |

#### Example: Advanced using multiple reclassargs

Create a new raster from the original where band 1,2,3 is converted to 1BB,4BUI, 4BUI respectively and reclassified. Note this uses the variadic `reclassarg` argument which can take as input an indefinite number of reclassargs (theoretically as many bands as you have)

```

UPDATE dummy_rast SET reclass_rast =
 ST_Reclass(rast,
 ROW(2,'0-87]:1-10, (87-100]:11-15, (101-254]:0-0', '4BUI',NULL)::reclassarg,
 ROW(1,'0-253]:1, 254:0', '1BB', NULL)::reclassarg,
 ROW(3,'0-70]:1, (70-86:2, [86-150]:3, [150-255:4', '4BUI', NULL)::reclassarg
) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast,1,i,j) As ov1, ST_Value(reclass_rast, 1, i, j) As ←
 rv1,
 ST_Value(rast,2,i,j) As ov2, ST_Value(reclass_rast, 2, i, j) As rv2,
 ST_Value(rast,3,i,j) As ov3, ST_Value(reclass_rast, 3, i, j) As rv3
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;

```

| col | row | ov1 | rv1 | ov2 | rv2 | ov3 | rv3 |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 1   | 1   | 253 | 1   | 78  | 9   | 70  | 1   |
| 2   | 1   | 254 | 0   | 98  | 14  | 86  | 3   |
| 3   | 1   | 253 | 1   | 122 | 0   | 100 | 3   |
| 1   | 2   | 253 | 1   | 96  | 14  | 80  | 2   |
| 2   | 2   | 254 | 0   | 118 | 0   | 108 | 3   |
| 3   | 2   | 254 | 0   | 180 | 0   | 162 | 4   |
| 1   | 3   | 250 | 1   | 99  | 15  | 90  | 3   |
| 2   | 3   | 254 | 0   | 112 | 0   | 108 | 3   |
| 3   | 3   | 254 | 0   | 169 | 0   | 175 | 4   |

### Example: Advanced Map a single band 32BF raster to multiple viewable bands

Create a new 3 band (8BUI,8BUI,8BUI viewable raster) from a raster that has only one 32bf band

```

ALTER TABLE wind ADD COLUMN rast_view raster;
UPDATE wind
 set rast_view = ST_AddBand(NULL,
 ARRAY[
 ST_Reclass(rast, 1, '0.1-10]:1-10,9-10]:11, (11-33:0'::text, '8BUI'::text,0),
 ST_Reclass(rast,1, '11-33):0-255, [0-32:0, (34-1000:0'::text, '8BUI'::text,0),
 ST_Reclass(rast,1, '0-32]:0, (32-100:100-255'::text, '8BUI'::text,0)
]
);

```

### See Also

[ST\\_AddBand](#), [ST\\_Band](#), [ST\\_BandPixelType](#), [ST\\_MakeEmptyRaster](#), [reclassarg](#), [ST\\_Value](#)

## 10.12.13 ST\_Union

**ST\_Union** — Returns the union of a set of raster tiles into a single raster composed of 1 or more bands.

### Synopsis

```

raster ST_Union(setof raster rast);
raster ST_Union(setof raster rast, unionarg[] unionargset);
raster ST_Union(setof raster rast, integer nband);
raster ST_Union(setof raster rast, text uniontype);
raster ST_Union(setof raster rast, integer nband, text uniontype);

```

## Description

Returns the union of a set of raster tiles into a single raster composed of at least one band. The resulting raster's extent is the extent of the whole set. In the case of intersection, the resulting value is defined by `uniontype` which is one of the following: LAST (default), FIRST, MIN, MAX, COUNT, SUM, MEAN, RANGE.



### Note

In order for rasters to be unioned, they must all have the same alignment. Use [ST\\_SameAlignment](#) and [ST\\_NotSameAlignmentReason](#) for more details and help. One way to fix alignment issues is to use [ST\\_Resample](#) and use the same reference raster for alignment.

Availability: 2.0.0

Enhanced: 2.1.0 Improved Speed (fully C-Based).

Availability: 2.1.0 `ST_Union(rast, unionarg)` variant was introduced.

Enhanced: 2.1.0 `ST_Union(rast)` (variant 1) unions all bands of all input rasters. Prior versions of PostGIS assumed the first band.

Enhanced: 2.1.0 `ST_Union(rast, uniontype)` (variant 4) unions all bands of all input rasters.

### Examples: Reconstitute a single band chunked raster tile

```
-- this creates a single band from first band of raster tiles
-- that form the original file system tile
SELECT filename, ST_Union(rast,1) As file_rast
FROM sometable WHERE filename IN('dem01', 'dem02') GROUP BY filename;
```

### Examples: Return a multi-band raster that is the union of tiles intersecting geometry

```
-- this creates a multi band raster collecting all the tiles that intersect a line
-- Note: In 2.0, this would have just returned a single band raster
-- , new union works on all bands by default
-- this is equivalent to unionarg: ARRAY[ROW(1, 'LAST'), ROW(2, 'LAST'), ROW(3, 'LAST')]:: ↵
unionarg[]
SELECT ST_Union(rast)
FROM aerials.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ↵
);
```

### Examples: Return a multi-band raster that is the union of tiles intersecting geometry

Here we use the longer syntax if we only wanted a subset of bands or we want to change order of bands

```
-- this creates a multi band raster collecting all the tiles that intersect a line
SELECT ST_Union(rast,ARRAY[ROW(2, 'LAST'), ROW(1, 'LAST'), ROW(3, 'LAST')]::unionarg[])
FROM aerials.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ↵
);
```

## See Also

[unionarg](#), [ST\\_Envelope](#), [ST\\_ConvexHull](#), [ST\\_Clip](#), [ST\\_Union](#)

## 10.13 Built-in Map Algebra Callback Functions

### 10.13.1 ST\_Distinct4ma

ST\_Distinct4ma — Raster processing function that calculates the number of unique pixel values in a neighborhood.

#### Synopsis

float8 **ST\_Distinct4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
double precision **ST\_Distinct4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

#### Description

Calculate the number of unique pixel values in a neighborhood of pixels.



**Note**  
Variant 1 is a specialized callback function for use as a callback parameter to [ST\\_MapAlgebraFctNgb](#).



**Note**  
Variant 2 is a specialized callback function for use as a callback parameter to [ST\\_MapAlgebra](#) (callback function version).



**Warning**  
Use of Variant 1 is discouraged since [ST\\_MapAlgebraFctNgb](#) has been deprecated as of 2.1.0.

Availability: 2.0.0  
Enhanced: 2.1.0 Addition of Variant 2

#### Examples

```
SELECT
 rid,
 st_value(
 st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_distinct4ma(float[],text,text[])':: ↵
 regprocedure, 'ignore', NULL), 2, 2
)
FROM dummy_rast
WHERE rid = 2;
 rid | st_value
-----+-----
 2 | 3
(1 row)
```

#### See Also

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra](#) (callback function version), [ST\\_Min4ma](#), [ST\\_Max4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)



### 10.13.2 ST\_InvDistWeight4ma

ST\_InvDistWeight4ma — Raster processing function that interpolates a pixel's value from the pixel's neighborhood.

#### Synopsis

double precision **ST\_InvDistWeight4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

#### Description

Calculate an interpolated value for a pixel using the Inverse Distance Weighted method.

There are two optional parameters that can be passed through `userargs`. The first parameter is the power factor (variable `k` in the equation below) between 0 and 1 used in the Inverse Distance Weighted equation. If not specified, default value is 1. The second parameter is the weight percentage applied only when the value of the pixel of interest is included with the interpolated value from the neighborhood. If not specified and the pixel of interest has a value, that value is returned.

The basic inverse distance weight equation is:

$$\hat{z}(x_o) = \frac{\sum_{j=1}^m z(x_j) d_{ij}^{-k}}{\sum_{j=1}^m d_{ij}^{-k}}$$

*k* = power factor, a real number between 0 and 1



#### Note

This function is a specialized callback function for use as a callback parameter to [ST\\_MapAlgebra \(callback function version\)](#).

Availability: 2.1.0

#### Examples

```
-- NEEDS EXAMPLE
```

#### See Also

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_MinDist4ma](#)

### 10.13.3 ST\_Max4ma

ST\_Max4ma — Raster processing function that calculates the maximum pixel value in a neighborhood.

#### Synopsis

float8 **ST\_Max4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);

double precision **ST\_Max4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calculate the maximum pixel value in a neighborhood of pixels.  
For Variant 2, a substitution value for NODATA pixels can be specified by passing that value to userargs.



**Note**  
Variant 1 is a specialized callback function for use as a callback parameter to [ST\\_MapAlgebraFctNgb](#).



**Note**  
Variant 2 is a specialized callback function for use as a callback parameter to [ST\\_MapAlgebra](#) (callback function version).



**Warning**  
Use of Variant 1 is discouraged since [ST\\_MapAlgebraFctNgb](#) has been deprecated as of 2.1.0.

Availability: 2.0.0  
Enhanced: 2.1.0 Addition of Variant 2

Examples

```
SELECT
 rid,
 st_value(
 st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_max4ma(float[][],text,text[])':: ↵
 regprocedure, 'ignore', NULL), 2, 2
)
FROM dummy_rast
WHERE rid = 2;
 rid | st_value
-----+-----
 2 | 254
(1 row)
```

See Also

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra](#) (callback function version), [ST\\_Min4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Range4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

10.13.4 ST\_Mean4ma

[ST\\_Mean4ma](#) — Raster processing function that calculates the mean pixel value in a neighborhood.

Synopsis

float8 **ST\_Mean4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
double precision **ST\_Mean4ma**(double precision[][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calculate the mean pixel value in a neighborhood of pixels.  
For Variant 2, a substitution value for NODATA pixels can be specified by passing that value to userargs.



**Note**  
Variant 1 is a specialized callback function for use as a callback parameter to `ST_MapAlgebraFctNgb`.



**Note**  
Variant 2 is a specialized callback function for use as a callback parameter to `ST_MapAlgebra` (callback function version).



**Warning**  
Use of Variant 1 is discouraged since `ST_MapAlgebraFctNgb` has been deprecated as of 2.1.0.

Availability: 2.0.0  
Enhanced: 2.1.0 Addition of Variant 2

Examples: Variant 1

```
SELECT
 rid,
 st_value(
 st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_mean4ma(float[][],text,text[])'::↔
 regprocedure, 'ignore', NULL), 2, 2
)
FROM dummy_rast
WHERE rid = 2;
rid | st_value
-----+-----
 2 | 253.222229003906
(1 row)
```

Examples: Variant 2

```
SELECT
 rid,
 st_value(
 ST_MapAlgebra(rast, 1, 'st_mean4ma(double precision[][][], integer[][], text ↔
 [])'::regprocedure,'32BF', 'FIRST', NULL, 1, 1)
 , 2, 2)
FROM dummy_rast
WHERE rid = 2;
rid | st_value
-----+-----
 2 | 253.222229003906
(1 row)
```

See Also

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra](#) (callback function version), [ST\\_Min4ma](#), [ST\\_Max4ma](#), [ST\\_Sum4ma](#), [ST\\_Range4ma](#), [ST\\_StdDev4ma](#)

10.13.5 ST\_Min4ma

ST\_Min4ma — Raster processing function that calculates the minimum pixel value in a neighborhood.

Synopsis

float8 **ST\_Min4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
double precision **ST\_Min4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calculate the minimum pixel value in a neighborhood of pixels.  
For Variant 2, a substitution value for NODATA pixels can be specified by passing that value to userargs.



**Note**  
Variant 1 is a specialized callback function for use as a callback parameter to [ST\\_MapAlgebraFctNgb](#).



**Note**  
Variant 2 is a specialized callback function for use as a callback parameter to [ST\\_MapAlgebra](#) (callback function version).



**Warning**  
Use of Variant 1 is discouraged since [ST\\_MapAlgebraFctNgb](#) has been deprecated as of 2.1.0.

Availability: 2.0.0  
Enhanced: 2.1.0 Addition of Variant 2

Examples

```
SELECT
 rid,
 st_value(
 st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_min4ma(float[][],text,text[])':: ↵
 regprocedure, 'ignore', NULL), 2, 2
)
FROM dummy_rast
WHERE rid = 2;
rid | st_value
-----+-----
 2 | 250
(1 row)
```

**See Also**

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra \(callback function version\)](#), [ST\\_Max4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Range4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

**10.13.6 ST\_MinDist4ma**

**ST\_MinDist4ma** — Raster processing function that returns the minimum distance (in number of pixels) between the pixel of interest and a neighboring pixel with value.

**Synopsis**

double precision **ST\_MinDist4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**Description**

Return the shortest distance (in number of pixels) between the pixel of interest and the closest pixel with value in the neighborhood.

**Note**

The intent of this function is to provide an informative data point that helps infer the usefulness of the pixel of interest's interpolated value from [ST\\_InvDistWeight4ma](#). This function is particularly useful when the neighborhood is sparsely populated.

**Note**

This function is a specialized callback function for use as a callback parameter to [ST\\_MapAlgebra \(callback function version\)](#).

Availability: 2.1.0

**Examples**

```
-- NEEDS EXAMPLE
```

**See Also**

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_InvDistWeight4ma](#)

**10.13.7 ST\_Range4ma**

**ST\_Range4ma** — Raster processing function that calculates the range of pixel values in a neighborhood.

**Synopsis**

float8 **ST\_Range4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
double precision **ST\_Range4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calculate the range of pixel values in a neighborhood of pixels.  
For Variant 2, a substitution value for NODATA pixels can be specified by passing that value to userargs.



**Note**  
Variant 1 is a specialized callback function for use as a callback parameter to [ST\\_MapAlgebraFctNgb](#).



**Note**  
Variant 2 is a specialized callback function for use as a callback parameter to [ST\\_MapAlgebra](#) (callback function version).



**Warning**  
Use of Variant 1 is discouraged since [ST\\_MapAlgebraFctNgb](#) has been deprecated as of 2.1.0.

Availability: 2.0.0  
Enhanced: 2.1.0 Addition of Variant 2

Examples

```
SELECT
 rid,
 st_value(
 st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_range4ma(float[][],text,text[])':: ↵
 regprocedure, 'ignore', NULL), 2, 2
)
FROM dummy_rast
WHERE rid = 2;
 rid | st_value
-----+-----
 2 | 4
(1 row)
```

See Also

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra](#) (callback function version), [ST\\_Min4ma](#), [ST\\_Max4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

10.13.8 ST\_StdDev4ma

ST\_StdDev4ma — Raster processing function that calculates the standard deviation of pixel values in a neighborhood.

Synopsis

float8 **ST\_StdDev4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
double precision **ST\_StdDev4ma**(double precision[][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calculate the standard deviation of pixel values in a neighborhood of pixels.



**Note**  
Variant 1 is a specialized callback function for use as a callback parameter to `ST_MapAlgebraFctNgb`.



**Note**  
Variant 2 is a specialized callback function for use as a callback parameter to `ST_MapAlgebra` (callback function version).



**Warning**  
Use of Variant 1 is discouraged since `ST_MapAlgebraFctNgb` has been deprecated as of 2.1.0.

Availability: 2.0.0  
Enhanced: 2.1.0 Addition of Variant 2

Examples

```
SELECT
 rid,
 st_value(
 st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_stddev4ma(float[],text,text[])':: ↵
 regprocedure, 'ignore', NULL), 2, 2
)
FROM dummy_rast
WHERE rid = 2;
rid | st_value
-----+-----
 2 | 1.30170822143555
(1 row)
```

See Also

`ST_MapAlgebraFctNgb`, `ST_MapAlgebra` (callback function version), `ST_Min4ma`, `ST_Max4ma`, `ST_Sum4ma`, `ST_Mean4ma`, `ST_Distinct4ma`, `ST_StdDev4ma`

10.13.9 ST\_Sum4ma

`ST_Sum4ma` — Raster processing function that calculates the sum of all pixel values in a neighborhood.

Synopsis

`float8 ST_Sum4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);`  
`double precision ST_Sum4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);`

Description

Calculate the sum of all pixel values in a neighborhood of pixels.

For Variant 2, a substitution value for NODATA pixels can be specified by passing that value to userargs.



**Note**  
Variant 1 is a specialized callback function for use as a callback parameter to [ST\\_MapAlgebraFctNgb](#).



**Note**  
Variant 2 is a specialized callback function for use as a callback parameter to [ST\\_MapAlgebra](#) (callback function version).



**Warning**  
Use of Variant 1 is discouraged since [ST\\_MapAlgebraFctNgb](#) has been deprecated as of 2.1.0.

Availability: 2.0.0

Enhanced: 2.1.0 Addition of Variant 2

Examples

```
SELECT
 rid,
 st_value(
 st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_sum4ma(float[][],text,text[])':: ↵
 regprocedure, 'ignore', NULL), 2, 2
)
FROM dummy_rast
WHERE rid = 2;
rid | st_value
-----+-----
 2 | 2279
(1 row)
```

See Also

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra](#) (callback function version), [ST\\_Min4ma](#), [ST\\_Max4ma](#), [ST\\_Mean4ma](#), [ST\\_Range4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

10.14 Raster Processing: DEM (Elevation)

10.14.1 ST\_Aspect

**ST\_Aspect** — Returns the aspect (in degrees by default) of an elevation raster band. Useful for analyzing terrain.





**Examples: Variant 2**

Complete example of tiles of a coverage. This query only works with PostgreSQL 9.1 or higher.

```
WITH foo AS (
 SELECT ST_Tile(
 ST_SetValues(
 ST_AddBand(
 ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
 1, '32BF', 0, -9999
),
 1, 1, 1, ARRAY[
 [1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 2, 1],
 [1, 2, 2, 3, 3, 1],
 [1, 1, 3, 2, 1, 1],
 [1, 2, 2, 1, 2, 1],
 [1, 1, 1, 1, 1, 1]
]::double precision[]
),
 2, 2
) AS rast
)
SELECT
 t1.rast,
 ST_Aspect(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;
```

**See Also**

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_TRI](#), [ST\\_TPI](#), [ST\\_Roughness](#), [ST\\_HillShade](#), [ST\\_Slope](#)

**10.14.2 ST\_HillShade**

**ST\_HillShade** — Returns the hypothetical illumination of an elevation raster band using provided azimuth, altitude, brightness and scale inputs.

**Synopsis**

raster **ST\_HillShade**(raster rast, integer band=1, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max\_bright=255, double precision scale=1.0, boolean interpolate\_nodata=FALSE);  
 raster **ST\_HillShade**(raster rast, integer band, raster customextent, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max\_bright=255, double precision scale=1.0, boolean interpolate\_nodata=FALSE);

**Description**

Returns the hypothetical illumination of an elevation raster band using the azimuth, altitude, brightness, and scale inputs. Utilizes map algebra and applies the hill shade equation to neighboring pixels. Return pixel values are between 0 and 255.

*azimuth* is a value between 0 and 360 degrees measured clockwise from North.

*altitude* is a value between 0 and 90 degrees where 0 degrees is at the horizon and 90 degrees is directly overhead.

*max\_bright* is a value between 0 and 255 with 0 as no brightness and 255 as max brightness.

scale is the ratio of vertical units to horizontal. For Feet:LatLon use scale=370400, for Meters:LatLon use scale=111120.

If `interpolate_nodata` is TRUE, values for NODATA pixels from the input raster will be interpolated using [ST\\_InvDistWeight4ma](#) before computing the hillshade illumination.



#### Note

For more information about Hillshade, please refer to [How hillshade works](#).

Availability: 2.0.0

Enhanced: 2.1.0 Uses `ST_MapAlgebra()` and added optional `interpolate_nodata` function parameter

Changed: 2.1.0 In prior versions, azimuth and altitude were expressed in radians. Now, azimuth and altitude are expressed in degrees

#### Examples: Variant 1

```
WITH foo AS (
 SELECT ST_SetValues(
 ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
 1, 1, 1, ARRAY[
 [1, 1, 1, 1, 1],
 [1, 2, 2, 2, 1],
 [1, 2, 3, 2, 1],
 [1, 2, 2, 2, 1],
 [1, 1, 1, 1, 1]
]::double precision[][])
) AS rast
)
SELECT
 ST_DumpValues(ST_Hillshade(rast, 1, '32BF'))
FROM foo
```

```

(1, "{ {NULL,NULL,NULL,NULL,NULL}, {NULL,251.32763671875,220.749786376953,147.224319458008, ←
 NULL}, {NULL,220.749786376953,180.312225341797,67.7497863769531,NULL}, {NULL ←
 ,147.224319458008
,67.7497863769531,43.1210060119629,NULL}, {NULL,NULL,NULL,NULL,NULL}} ")
(1 row)
```

#### Examples: Variant 2

Complete example of tiles of a coverage. This query only works with PostgreSQL 9.1 or higher.

```
WITH foo AS (
 SELECT ST_Tile(
 ST_SetValues(
 ST_AddBand(
 ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
 1, '32BF', 0, -9999
),
 1, 1, 1, ARRAY[
```

```

 [1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 2, 1],
 [1, 2, 2, 3, 3, 1],
 [1, 1, 3, 2, 1, 1],
 [1, 2, 2, 1, 2, 1],
 [1, 1, 1, 1, 1, 1]
]::double precision[]
),
 2, 2
) AS rast
)
SELECT
 t1.rast,
 ST_Hillshade(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;

```

**See Also**

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_TRI](#), [ST\\_TPI](#), [ST\\_Roughness](#), [ST\\_Aspect](#), [ST\\_Slope](#)

**10.14.3 ST\_Roughness**

**ST\_Roughness** — Returns a raster with the calculated "roughness" of a DEM.

**Synopsis**

```
raster ST_Roughness(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);
```

**Description**

Calculates the "roughness" of a DEM, by subtracting the maximum from the minimum for a given area.

Availability: 2.1.0

**Examples**

```
-- needs examples
```

**See Also**

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_TRI](#), [ST\\_TPI](#), [ST\\_Slope](#), [ST\\_HillShade](#), [ST\\_Aspect](#)

**10.14.4 ST\_Slope**

**ST\_Slope** — Returns the slope (in degrees by default) of an elevation raster band. Useful for analyzing terrain.

## Synopsis

raster **ST\_Slope**(raster rast, integer nband=1, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean interpolate\_nodata=FALSE);

raster **ST\_Slope**(raster rast, integer nband, raster customextent, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean interpolate\_nodata=FALSE);

## Description

Returns the slope (in degrees by default) of an elevation raster band. Utilizes map algebra and applies the slope equation to neighboring pixels.

`units` indicates the units of the slope. Possible values are: RADIANS, DEGREES (default), PERCENT.

`scale` is the ratio of vertical units to horizontal. For Feet:LatLon use `scale=370400`, for Meters:LatLon use `scale=111120`.

If `interpolate_nodata` is TRUE, values for NODATA pixels from the input raster will be interpolated using [ST\\_InvDistWeight4ma](#) before computing the surface slope.



### Note

For more information about Slope, Aspect and Hillshade, please refer to [ESRI - How hillshade works](#) and [ERDAS Field Guide - Slope Images](#).

Availability: 2.0.0

Enhanced: 2.1.0 Uses `ST_MapAlgebra()` and added optional `units`, `scale`, `interpolate_nodata` function parameters

Changed: 2.1.0 In prior versions, return values were in radians. Now, return values default to degrees

## Examples: Variant 1

```
WITH foo AS (
 SELECT ST_SetValues(
 ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
 1, 1, 1, ARRAY[
 [1, 1, 1, 1, 1],
 [1, 2, 2, 2, 1],
 [1, 2, 3, 2, 1],
 [1, 2, 2, 2, 1],
 [1, 1, 1, 1, 1]
]::double precision[][])
) AS rast
)
SELECT
 ST_DumpValues(ST_Slope(rast, 1, '32BF'))
FROM foo

 st_dumpvalues
```

```
(1, "{10.0249881744385,21.5681285858154,26.5650520324707,21.5681285858154,10.0249881744385},{21.5681285858154,26.5650520324707,36.8698959350586,0,36.8698959350586,26.5650520324707},{21.5681285858154,35.26438905681285858154,26.5650520324707,21.5681285858154,10.0249881744385}}")
(1 row)
```

**Examples: Variant 2**

Complete example of tiles of a coverage. This query only works with PostgreSQL 9.1 or higher.

```
WITH foo AS (
 SELECT ST_Tile(
 ST_SetValues(
 ST_AddBand(
 ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
 1, '32BF', 0, -9999
),
 1, 1, 1, ARRAY[
 [1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 2, 1],
 [1, 2, 2, 3, 3, 1],
 [1, 1, 3, 2, 1, 1],
 [1, 2, 2, 1, 2, 1],
 [1, 1, 1, 1, 1, 1]
]::double precision[]
),
 2, 2
) AS rast
)
SELECT
 t1.rast,
 ST_Slope(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;
```

**See Also**

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_TRI](#), [ST\\_TPI](#), [ST\\_Roughness](#), [ST\\_HillShade](#), [ST\\_Aspect](#)

**10.14.5 ST\_TPI**

**ST\_TPI** — Returns a raster with the calculated Topographic Position Index.

**Synopsis**

raster **ST\_TPI**(raster rast, integer nband, raster customextent, text pixeltype='32BF' , boolean interpolate\_nodata=FALSE );

**Description**

Calculates the Topographic Position Index, which is defined as the focal mean with radius of one minus the center cell.

**Note**

This function only supports a focalmean radius of one.

Availability: 2.1.0

## Examples

```
-- needs examples
```

## See Also

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_TRI](#), [ST\\_Roughness](#), [ST\\_Slope](#), [ST\\_HillShade](#), [ST\\_Aspect](#)

### 10.14.6 ST\_TRI

**ST\_TRI** — Returns a raster with the calculated Terrain Ruggedness Index.

## Synopsis

raster **ST\_TRI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate\_nodata=FALSE );

## Description

Terrain Ruggedness Index is calculated by comparing a central pixel with its neighbors, taking the absolute values of the differences, and averaging the result.



### Note

This function only supports a focalmean radius of one.

Availability: 2.1.0

## Examples

```
-- needs examples
```

## See Also

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_Roughness](#), [ST\\_TPI](#), [ST\\_Slope](#), [ST\\_HillShade](#), [ST\\_Aspect](#)

## 10.15 Raster Processing: Raster to Geometry

### 10.15.1 Box3D

**Box3D** — Returns the box 3d representation of the enclosing box of the raster.

## Synopsis

box3d **Box3D**(raster rast);

**Description**

Returns the box representing the extent of the raster.

The polygon is defined by the corner points of the bounding box ((MINX, MINY), (MAXX, MAXY))

Changed: 2.0.0 In pre-2.0 versions, there used to be a box2d instead of box3d. Since box2d is a deprecated type, this was changed to box3d.

**Examples**

```
SELECT
 rid,
 Box3D(rast) AS rastbox
FROM dummy_rast;
```

| rid | rastbox                                         |
|-----|-------------------------------------------------|
| 1   | BOX3D(0.5 0.5 0,20.5 60.5 0)                    |
| 2   | BOX3D(3427927.75 5793243.5 0,3427928 5793244 0) |

**See Also**

[ST\\_Envelope](#)

**10.15.2 ST\_ConvexHull**

ST\_ConvexHull — Return the convex hull geometry of the raster including pixel values equal to BandNoDataValue. For regular shaped and non-skewed rasters, this gives the same result as ST\_Envelope so only useful for irregularly shaped or skewed rasters.

**Synopsis**

geometry **ST\_ConvexHull**(raster rast);

**Description**

Return the convex hull geometry of the raster including the NoDataBandValue band pixels. For regular shaped and non-skewed rasters, this gives more or less the same result as ST\_Envelope so only useful for irregularly shaped or skewed rasters.



**Note**  
ST\_Envelope floors the coordinates and hence add a little buffer around the raster so the answer is subtly different from ST\_ConvexHull which does not floor.

**Examples**

Refer to [PostGIS Raster Specification](#) for a diagram of this.

```
-- Note envelope and convexhull are more or less the same
SELECT ST_AsText(ST_ConvexHull(rast)) As convexhull,
 ST_AsText(ST_Envelope(rast)) As env
FROM dummy_rast WHERE rid=1;
```

| convexhull | env |
|------------|-----|
|------------|-----|



```
-----+----- ↵
POLYGON((0.5 0.5,20.5 0.5,20.5 60.5,0.5 60.5,0.5 0.5)) | POLYGON((0 0,20 0,20 60,0 60,0 0) ↵
)

-- now we skew the raster
-- note how the convex hull and envelope are now different
SELECT ST_AsText(ST_ConvexHull(rast)) As convhull,
 ST_AsText(ST_Envelope(rast)) As env
FROM (SELECT ST_SetRotation(rast, 0.1, 0.1) As rast
 FROM dummy_rast WHERE rid=1) As foo;

 convhull | env
-----+----- ↵
POLYGON((0.5 0.5,20.5 1.5,22.5 61.5,2.5 60.5,0.5 0.5)) | POLYGON((0 0,22 0,22 61,0 61,0 0) ↵
)
```

**See Also**

[ST\\_Envelope](#), [ST\\_MinConvexHull](#), [ST\\_ConvexHull](#), [ST\\_AsText](#)

**10.15.3 ST\_DumpAsPolygons**

**ST\_DumpAsPolygons** — Returns a set of geomval (geom,val) rows, from a given raster band. If no band number is specified, band num defaults to 1.

**Synopsis**

setof geomval **ST\_DumpAsPolygons**(raster rast, integer band\_num=1, boolean exclude\_nodata\_value=TRUE);

**Description**

This is a set-returning function (SRF). It returns a set of geomval rows, formed by a geometry (geom) and a pixel band value (val). Each polygon is the union of all pixels for that band that have the same pixel value denoted by val.

ST\_DumpAsPolygon is useful for polygonizing rasters. It is the reverse of a GROUP BY in that it creates new rows. For example it can be used to expand a single raster into multiple POLYGONS/MULTIPOLYGONS.

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

Availability: Requires GDAL 1.7 or higher.



**Note**  
If there is a no data value set for a band, pixels with that value will not be returned except in the case of exclude\_nodata\_value=false.



**Note**  
If you only care about count of pixels with a given value in a raster, it is faster to use [ST\\_ValueCount](#).



**Note**  
This is different than ST\_PixelAsPolygons where one geometry is returned for each pixel regardless of pixel value.

Examples

```
-- this syntax requires PostgreSQL 9.3+
SELECT val, ST_AsText(geom) As geomwkt
FROM (
 SELECT dp.*
 FROM dummy_rast, LATERAL ST_DumpAsPolygons(rast) AS dp
 WHERE rid = 2
) As foo
WHERE val BETWEEN 249 and 251
ORDER BY val;
```

| val | geomwkt                                                                                                              |
|-----|----------------------------------------------------------------------------------------------------------------------|
| 249 | POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 5793243.85,3427928 5793243.95,3427927.95 5793243.95))   |
| 250 | POLYGON((3427927.75 5793243.9,3427927.75 5793243.85,3427927.8 5793243.85,3427927.8 5793243.9,3427927.75 5793243.9))  |
| 250 | POLYGON((3427927.8 5793243.8,3427927.8 5793243.75,3427927.85 5793243.75,3427927.85 5793243.8,3427927.8 5793243.8))   |
| 251 | POLYGON((3427927.75 5793243.85,3427927.75 5793243.8,3427927.8 5793243.8,3427927.8 5793243.85,3427927.75 5793243.85)) |

See Also

[geomval](#), [ST\\_Value](#), [ST\\_Polygon](#), [ST\\_ValueCount](#)

10.15.4 ST\_Envelope

ST\_Envelope — Returns the polygon representation of the extent of the raster.

Synopsis

geometry **ST\_Envelope**(raster rast);

Description

Returns the polygon representation of the extent of the raster in spatial coordinate units defined by srid. It is a float8 minimum bounding box represented as a polygon.

The polygon is defined by the corner points of the bounding box ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY))

Examples

```
SELECT rid, ST_AsText(ST_Envelope(rast)) As envgeomwkt
FROM dummy_rast;
```

| rid | envgeomwkt                                                                                 |
|-----|--------------------------------------------------------------------------------------------|
| 1   | POLYGON((0 0,20 0,20 60,0 60,0 0))                                                         |
| 2   | POLYGON((3427927 5793243,3427928 5793243,3427928 5793244,3427927 5793244,3427927 5793243)) |

**See Also**

[ST\\_Envelope](#), [ST\\_AsText](#), [ST\\_SRID](#)

**10.15.5 ST\_MinConvexHull**

**ST\_MinConvexHull** — Return the convex hull geometry of the raster excluding NODATA pixels.

**Synopsis**

geometry **ST\_MinConvexHull**(raster rast, integer nband=NULL);

**Description**

Return the convex hull geometry of the raster excluding NODATA pixels. If *nband* is NULL, all bands of the raster are considered.

Availability: 2.1.0

**Examples**

```
WITH foo AS (
 SELECT
 ST_SetValues(
 ST_SetValues(
 ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(9, 9, 0, 0, 1, -1, 0, 0, 0), 1, '8 ←
 BUI', 0, 0), 2, '8BUI', 1, 0),
 1, 1, 1,
 ARRAY[
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0, 0, 1],
 [0, 0, 0, 1, 1, 0, 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0]
]::double precision[][])
),
 2, 1, 1,
 ARRAY[
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [1, 0, 0, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 1, 1, 0, 0, 0],
 [0, 0, 0, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 0]
]::double precision[][])
) AS rast
)
SELECT
 ST_AsText(ST_ConvexHull(rast)) AS hull,
 ST_AsText(ST_MinConvexHull(rast)) AS mhull,
```

```

 ST_AsText(ST_MinConvexHull(rast, 1)) AS mhull_1,
 ST_AsText(ST_MinConvexHull(rast, 2)) AS mhull_2
FROM foo

```

|                   |  |         |  |   |
|-------------------|--|---------|--|---|
| hull              |  | mhull   |  | ↔ |
| mhull_1           |  | mhull_2 |  |   |
| -----+-----+----- |  |         |  |   |

```

POLYGON((0 0,9 0,9 -9,0 -9,0 0)) | POLYGON((0 -3,9 -3,9 -9,0 -9,0 -3)) | POLYGON((3 -3,9 -3,9 -6,3 -6,3 -3)) | POLYGON((0 -3,6 -3,6 -9,0 -9,0 -3))

```

## See Also

[ST\\_Envelope](#), [ST\\_ConvexHull](#), [ST\\_ConvexHull](#), [ST\\_AsText](#)

## 10.15.6 ST\_Polygon

**ST\_Polygon** — Returns a multipolygon geometry formed by the union of pixels that have a pixel value that is not no data value. If no band number is specified, band num defaults to 1.

### Synopsis

geometry **ST\_Polygon**(raster rast, integer band\_num=1);

### Description

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

Availability: 0.1.6 Requires GDAL 1.7 or higher.

Enhanced: 2.1.0 Improved Speed (fully C-Based) and the returning multipolygon is ensured to be valid.

Changed: 2.1.0 In prior versions would sometimes return a polygon, changed to always return multipolygon.

### Examples

```

-- by default no data band value is 0 or not set, so polygon will return a square polygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt

MULTIPOLYGON(((3427927.75 5793244,3427928 5793244,3427928 5793243.75,3427927.75 5793243.75,3427927.75 5793244)))

-- now we change the no data value of first band
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,1,254)
WHERE rid = 2;
SELECT rid, ST_BandNoDataValue(rast)
from dummy_rast where rid = 2;

-- ST_Polygon excludes the pixel value 254 and returns a multipolygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

```

```

geomwkt

MULTIPOLYGON(((3427927.9 5793243.95,3427927.85 5793243.95,3427927.85 5793244,3427927.9 ↵
5793244,3427927.9 5793243.95)),((3427928 5793243.85,3427928 5793243.8,3427927.95 ↵
5793243.8,3427927.95 5793243.85,3427927.9 5793243.85,3427927.9 5793243.9,3427927.9 ↵
5793243.95,3427927.95 5793243.95,3427928 5793243.95,3427928 5793243.85)),((3427927.8 ↵
5793243.75,3427927.75 5793243.75,3427927.75 5793243.8,3427927.75 5793243.85,3427927.75 ↵
5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.9,3427927.8 ↵
5793243.85,3427927.85 5793243.85,3427927.85 5793243.8,3427927.85 5793243.75,3427927.8 ↵
5793243.75)))

-- Or if you want the no data value different for just one time

SELECT ST_AsText (
 ST_Polygon (
 ST_SetBandNoDataValue (rast,1,252)
)
) As geomwkt
FROM dummy_rast
WHERE rid =2;

geomwkt

MULTIPOLYGON(((3427928 5793243.85,3427928 5793243.8,3427928 5793243.75,3427927.85 ↵
5793243.75,3427927.8 5793243.75,3427927.8 5793243.8,3427927.75 5793243.8,3427927.75 ↵
5793243.85,3427927.75 5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.85 ↵
5793244,3427927.9 5793244,3427928 5793244,3427928 5793243.95,3427928 5793243.85) ↵
,(3427927.9 5793243.9,3427927.9 5793243.85,3427927.95 5793243.85,3427927.95 ↵
5793243.9,3427927.9 5793243.9)))

```

**See Also**

[ST\\_Value](#), [ST\\_DumpAsPolygons](#)

## 10.16 Raster Operators

### 10.16.1 &&

**&&** — Returns TRUE if A's bounding box intersects B's bounding box.

**Synopsis**

```

boolean &&(raster A , raster B);
boolean &&(raster A , geometry B);
boolean &&(geometry B , raster A);

```

**Description**

The **&&** operator returns TRUE if the bounding box of raster/geometr A intersects the bounding box of raster/geometr B.

**Note**

This operand will make use of any indexes that may be available on the rasters.

Availability: 2.0.0

Examples

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast && B.rast As intersect
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B LIMIT 3;
```

| a_rid | b_rid | intersect |
|-------|-------|-----------|
| 2     | 2     | t         |
| 2     | 3     | f         |
| 2     | 1     | f         |

10.16.2 &<

&< — Returns TRUE if A’s bounding box is to the left of B’s.

Synopsis

boolean &<( raster A , raster B );

Description

The &< operator returns TRUE if the bounding box of raster A overlaps or is to the left of the bounding box of raster B, or more accurately, overlaps or is NOT to the right of the bounding box of raster B.



**Note**  
This operand will make use of any indexes that may be available on the rasters.

Examples

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &< B.rast As overleft
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

| a_rid | b_rid | overleft |
|-------|-------|----------|
| 2     | 2     | t        |
| 2     | 3     | f        |
| 2     | 1     | f        |
| 3     | 2     | t        |
| 3     | 3     | t        |
| 3     | 1     | f        |
| 1     | 2     | t        |
| 1     | 3     | t        |
| 1     | 1     | t        |

10.16.3 &>

&> — Returns TRUE if A’s bounding box is to the right of B’s.

Synopsis

boolean `&>( raster A , raster B );`

Description

The `&>` operator returns `TRUE` if the bounding box of raster A overlaps or is to the right of the bounding box of raster B, or more accurately, overlaps or is `NOT` to the left of the bounding box of raster B.



Note

This operand will make use of any indexes that may be available on the geometries.

Examples

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &> B.rast As overright
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

| a_rid | b_rid | overright |
|-------|-------|-----------|
| 2     | 2     | t         |
| 2     | 3     | t         |
| 2     | 1     | t         |
| 3     | 2     | f         |
| 3     | 3     | t         |
| 3     | 1     | f         |
| 1     | 2     | f         |
| 1     | 3     | t         |
| 1     | 1     | t         |

10.16.4 =

`=` — Returns `TRUE` if A's bounding box is the same as B's. Uses double precision bounding box.

Synopsis

boolean `=( raster A , raster B );`

Description

The `=` operator returns `TRUE` if the bounding box of raster A is the same as the bounding box of raster B. PostgreSQL uses the `=`, `<`, and `>` operators defined for rasters to perform internal orderings and comparison of rasters (ie. in a `GROUP BY` or `ORDER BY` clause).



Caution

This operand will `NOT` make use of any indexes that may be available on the rasters. Use `~=` instead. This operator exists mostly so one can group by the raster column.

Availability: 2.1.0

**See Also**

~=

**10.16.5 @**

@ — Returns TRUE if A's bounding box is contained by B's. Uses double precision bounding box.

**Synopsis**

```
boolean @(raster A , raster B);
boolean @(geometry A , raster B);
boolean @(raster B , geometry A);
```

**Description**

The @ operator returns TRUE if the bounding box of raster/geometry A is contained by bounding box of raster/geometr B.

**Note**

This operand will use spatial indexes on the rasters.

Availability: 2.0.0 raster @ raster, raster @ geometry introduced

Availability: 2.0.5 geometry @ raster introduced

**See Also**

~

**10.16.6 ~=**

~= — Returns TRUE if A's bounding box is the same as B's.

**Synopsis**

```
boolean ~= (raster A , raster B);
```

**Description**

The ~= operator returns TRUE if the bounding box of raster A is the same as the bounding box of raster B.

**Note**

This operand will make use of any indexes that may be available on the rasters.

Availability: 2.0.0



## Examples

Very useful usecase is for taking two sets of single band rasters that are of the same chunk but represent different themes and creating a multi-band raster

```
SELECT ST_AddBand(prec.rast, alt.rast) As new_rast
FROM prec INNER JOIN alt ON (prec.rast ~= alt.rast);
```

## See Also

[ST\\_AddBand](#), [=](#)

## 10.16.7 ~

~ — Returns TRUE if A's bounding box is contains B's. Uses double precision bounding box.

## Synopsis

```
boolean ~(raster A , raster B);
boolean ~(geometry A , raster B);
boolean ~(raster B , geometry A);
```

## Description

The ~ operator returns TRUE if the bounding box of raster/geometry A is contains bounding box of raster/geometr B.



### Note

This operand will use spatial indexes on the rasters.

Availability: 2.0.0

## See Also

[@](#)

# 10.17 Raster and Raster Band Spatial Relationships

## 10.17.1 ST\_Contains

ST\_Contains — Return true if no points of raster rastB lie in the exterior of raster rastA and at least one point of the interior of rastB lies in the interior of rastA.

## Synopsis

```
boolean ST_Contains(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean ST_Contains(raster rastA , raster rastB);
```

Description

Raster `rastA` contains `rastB` if and only if no points of `rastB` lie in the exterior of `rastA` and at least one point of the interior of `rastB` lies in the interior of `rastA`. If the band number is not provided (or set to `NULL`), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not `NODATA`) are considered in the test.



**Note**  
This function will make use of any indexes that may be available on the rasters.



**Note**  
To test the spatial relationship of a raster and a geometry, use `ST_Polygon` on the raster, e.g. `ST_Contains(ST_Polygon(raster), geometry)` or `ST_Contains(geometry, ST_Polygon(raster))`.



**Note**  
`ST_Contains()` is the inverse of `ST_Within()`. So, `ST_Contains(rastA, rastB)` implies `ST_Within(rastB, rastA)`.

Availability: 2.1.0

Examples

```
-- specified band numbers
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↵
 dummy_rast r2 WHERE r1.rid = 1;

NOTICE: The first raster provided has no bands
 rid | rid | st_contains
-----+-----+-----
 1 | 1 |
 1 | 2 | f

-- no band numbers specified
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ↵
 dummy_rast r2 WHERE r1.rid = 1;
 rid | rid | st_contains
-----+-----+-----
 1 | 1 | t
 1 | 2 | f
```

See Also

[ST\\_Intersects](#), [ST\\_Within](#)

10.17.2 ST\_ContainsProperly

`ST_ContainsProperly` — Return true if `rastB` intersects the interior of `rastA` but not the boundary or exterior of `rastA`.

## Synopsis

boolean **ST\_ContainsProperly**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
 boolean **ST\_ContainsProperly**( raster rastA , raster rastB );

## Description

Raster rastA contains properly rastB if rastB intersects the interior of rastA but not the boundary or exterior of rastA. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.

Raster rastA does not contain properly itself but does contain itself.



### Note

This function will make use of any indexes that may be available on the rasters.



### Note

To test the spatial relationship of a raster and a geometry, use ST\_Polygon on the raster, e.g. ST\_ContainsProperly(ST\_Polygon(raster), geometry) or ST\_ContainsProperly(geometry, ST\_Polygon(raster)).

Availability: 2.1.0

## Examples

```
SELECT r1.rid, r2.rid, ST_ContainsProperly(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
 dummy_rast r2 WHERE r1.rid = 2;
```

| rid | rid | st_containsproperly |
|-----|-----|---------------------|
| 2   | 1   | f                   |
| 2   | 2   | f                   |

## See Also

[ST\\_Intersects](#), [ST\\_Contains](#)

### 10.17.3 ST\_Covers

**ST\_Covers** — Return true if no points of raster rastB lie outside raster rastA.

## Synopsis

boolean **ST\_Covers**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
 boolean **ST\_Covers**( raster rastA , raster rastB );

Description

Raster `rastA` covers `rastB` if and only if no points of `rastB` lie in the exterior of `rastA`. If the band number is not provided (or set to `NULL`), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not `NODATA`) are considered in the test.



**Note**  
This function will make use of any indexes that may be available on the rasters.



**Note**  
To test the spatial relationship of a raster and a geometry, use `ST_Polygon` on the raster, e.g. `ST_Covers(ST_Polygon(raster), geometry)` or `ST_Covers(geometry, ST_Polygon(raster))`.

Availability: 2.1.0

Examples

```
SELECT r1.rid, r2.rid, ST_Covers(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↔
dummy_rast r2 WHERE r1.rid = 2;
```

| rid | rid | st_covers |
|-----|-----|-----------|
| 2   | 1   | f         |
| 2   | 2   | t         |

See Also

[ST\\_Intersects](#), [ST\\_CoveredBy](#)

10.17.4 ST\_CoveredBy

`ST_CoveredBy` — Return true if no points of raster `rastA` lie outside raster `rastB`.

Synopsis

boolean **ST\_CoveredBy**( raster `rastA` , integer `nbandA` , raster `rastB` , integer `nbandB` );  
boolean **ST\_CoveredBy**( raster `rastA` , raster `rastB` );

Description

Raster `rastA` is covered by `rastB` if and only if no points of `rastA` lie in the exterior of `rastB`. If the band number is not provided (or set to `NULL`), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not `NODATA`) are considered in the test.



**Note**  
This function will make use of any indexes that may be available on the rasters.



**Note**  
To test the spatial relationship of a raster and a geometry, use ST\_Polygon on the raster, e.g. ST\_CoveredBy(ST\_Polygon(raster), geometry) or ST\_CoveredBy(geometry, ST\_Polygon(raster)).

Availability: 2.1.0

Examples

```
SELECT r1.rid, r2.rid, ST_CoveredBy(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
dummy_rast r2 WHERE r1.rid = 2;
```

| rid | rid | st_coveredby |
|-----|-----|--------------|
| 2   | 1   | f            |
| 2   | 2   | t            |

See Also

[ST\\_Intersects](#), [ST\\_Covers](#)

10.17.5 ST\_Disjoint

ST\_Disjoint — Return true if raster rastA does not spatially intersect rastB.

Synopsis

boolean **ST\_Disjoint**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean **ST\_Disjoint**( raster rastA , raster rastB );

Description

Raster rastA and rastB are disjointed if they do not share any space together. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.



**Note**  
This function does NOT use any indexes.



**Note**  
To test the spatial relationship of a raster and a geometry, use ST\_Polygon on the raster, e.g. ST\_Disjoint(ST\_Polygon(raster), geometry).

Availability: 2.1.0

Examples

```
-- rid = 1 has no bands, hence the NOTICE and the NULL value for st_disjoint
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↵
 dummy_rast r2 WHERE r1.rid = 2;

NOTICE: The second raster provided has no bands
 rid | rid | st_disjoint
-----+-----+-----
 2 | 1 |
 2 | 2 | f
```

```
-- this time, without specifying band numbers
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ↵
 dummy_rast r2 WHERE r1.rid = 2;

 rid | rid | st_disjoint
-----+-----+-----
 2 | 1 | t
 2 | 2 | f
```

See Also

ST\_Intersects

10.17.6 ST\_Intersects

ST\_Intersects — Return true if raster rastA spatially intersects raster rastB.

Synopsis

boolean **ST\_Intersects**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean **ST\_Intersects**( raster rastA , raster rastB );  
boolean **ST\_Intersects**( raster rast , integer nband , geometry geommin );  
boolean **ST\_Intersects**( raster rast , geometry geommin , integer nband=NULL );  
boolean **ST\_Intersects**( geometry geommin , raster rast , integer nband=NULL );

Description

Return true if raster rastA spatially intersects raster rastB. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.



**Note**  
This function will make use of any indexes that may be available on the rasters.

Enhanced: 2.0.0 support raster/raster intersects was introduced.



**Warning**  
Changed: 2.1.0 The behavior of the ST\_Intersects(raster, geometry) variants changed to match that of ST\_Intersects(geometry, raster).

## Examples

```
-- different bands of same raster
SELECT ST_Intersects(rast, 2, rast, 3) FROM dummy_rast WHERE rid = 2;

 st_intersects

t
```

## See Also

[ST\\_Intersection](#), [ST\\_Disjoint](#)

## 10.17.7 ST\_Overlaps

**ST\_Overlaps** — Return true if raster rastA and rastB intersect but one does not completely contain the other.

### Synopsis

boolean **ST\_Overlaps**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
 boolean **ST\_Overlaps**( raster rastA , raster rastB );

### Description

Return true if raster rastA spatially overlaps raster rastB. This means that rastA and rastB intersect but one does not completely contain the other. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.



#### Note

This function will make use of any indexes that may be available on the rasters.



#### Note

To test the spatial relationship of a raster and a geometry, use `ST_Polygon` on the raster, e.g. `ST_Overlaps(ST_Polygon(raster), geometry)`.

Availability: 2.1.0

## Examples

```
-- comparing different bands of same raster
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;

 st_overlaps

f
```

See Also

[ST\\_Intersects](#)

10.17.8 ST\_Touches

ST\_Touches — Return true if raster rastA and rastB have at least one point in common but their interiors do not intersect.

Synopsis

boolean **ST\_Touches**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean **ST\_Touches**( raster rastA , raster rastB );

Description

Return true if raster rastA spatially touches raster rastB. This means that rastA and rastB have at least one point in common but their interiors do not intersect. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.



**Note**  
This function will make use of any indexes that may be available on the rasters.



**Note**  
To test the spatial relationship of a raster and a geometry, use ST\_Polygon on the raster, e.g. ST\_Touches(ST\_Polygon(raster), geometry).

Availability: 2.1.0

Examples

```
SELECT r1.rid, r2.rid, ST_Touches(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↔
dummy_rast r2 WHERE r1.rid = 2;
```

| rid | rid | st_touches |
|-----|-----|------------|
| 2   | 1   | f          |
| 2   | 2   | f          |

See Also

[ST\\_Intersects](#)

10.17.9 ST\_SameAlignment

ST\_SameAlignment — Returns true if rasters have same skew, scale, spatial ref, and offset (pixels can be put on same grid without cutting into pixels) and false if they don't with notice detailing issue.



## Synopsis

boolean **ST\_SameAlignment**( raster rastA , raster rastB );  
 boolean **ST\_SameAlignment**( double precision ulx1 , double precision uly1 , double precision scalex1 , double precision scaley1 , double precision skewx1 , double precision skewy1 , double precision ulx2 , double precision uly2 , double precision scalex2 , double precision scaley2 , double precision skewx2 , double precision skewy2 );  
 boolean **ST\_SameAlignment**( raster set rastfield );

## Description

Non-Aggregate version (Variants 1 and 2): Returns true if the two rasters (either provided directly or made using the values for upperleft, scale, skew and srid) have the same scale, skew, srid and at least one of any of the four corners of any pixel of one raster falls on any corner of the grid of the other raster. Returns false if they don't and a NOTICE detailing the alignment issue.

Aggregate version (Variant 3): From a set of rasters, returns true if all rasters in the set are aligned. The ST\_SameAlignment() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do.

Availability: 2.0.0

Enhanced: 2.1.0 addition of Aggregate variant

## Examples: Rasters

```
SELECT ST_SameAlignment (
 ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
 ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0)
) as sm;

sm

t
```

```
SELECT ST_SameAlignment(A.rast,b.rast)
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;

NOTICE: The two rasters provided have different SRIDs
NOTICE: The two rasters provided have different SRIDs
 st_samealignment

t
f
f
f
```

## See Also

Section 9.1, [ST\\_NotSameAlignmentReason](#), [ST\\_MakeEmptyRaster](#)

### 10.17.10 ST\_NotSameAlignmentReason

ST\_NotSameAlignmentReason — Returns text stating if rasters are aligned and if not aligned, a reason why.

## Synopsis

text **ST\_NotSameAlignmentReason**(raster rastA, raster rastB);

Description

Returns text stating if rasters are aligned and if not aligned, a reason why.



**Note**  
If there are several reasons why the rasters are not aligned, only one reason (the first test to fail) will be returned.

Availability: 2.1.0

Examples

```
SELECT
 ST_SameAlignment (
 ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
 ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
),
 ST_NotSameAlignmentReason (
 ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
 ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
)
;

st_samealignment | st_otsamealignmentreason
-----+-----
f | The rasters have different scales on the X axis
(1 row)
```

See Also

Section 9.1, [ST\\_SameAlignment](#)

10.17.11 ST\_Within

ST\_Within — Return true if no points of raster rastA lie in the exterior of raster rastB and at least one point of the interior of rastA lies in the interior of rastB.

Synopsis

boolean **ST\_Within**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean **ST\_Within**( raster rastA , raster rastB );

Description

Raster rastA is within rastB if and only if no points of rastA lie in the exterior of rastB and at least one point of the interior of rastA lies in the interior of rastB. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.



**Note**  
This operand will make use of any indexes that may be available on the rasters.



**Note**  
To test the spatial relationship of a raster and a geometry, use `ST_Polygon` on the raster, e.g. `ST_Within(ST_Polygon(raster), geometry)` or `ST_Within(geometry, ST_Polygon(raster))`.



**Note**  
`ST_Within()` is the inverse of `ST_Contains()`. So, `ST_Within(rastA, rastB)` implies `ST_Contains(rastB, rastA)`.

Availability: 2.1.0

Examples

```
SELECT r1.rid, r2.rid, ST_Within(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↔
dummy_rast r2 WHERE r1.rid = 2;
```

| rid | rid | st_within |
|-----|-----|-----------|
| 2   | 1   | f         |
| 2   | 2   | t         |

See Also

[ST\\_Intersects](#), [ST\\_Contains](#), [ST\\_DWithin](#), [ST\\_DFullyWithin](#)

10.17.12 ST\_DWithin

`ST_DWithin` — Return true if rasters `rastA` and `rastB` are within the specified distance of each other.

Synopsis

boolean **ST\_DWithin**( raster `rastA` , integer `nbandA` , raster `rastB` , integer `nbandB` , double precision `distance_of_srid` );  
boolean **ST\_DWithin**( raster `rastA` , raster `rastB` , double precision `distance_of_srid` );

Description

Return true if rasters `rastA` and `rastB` are within the specified distance of each other. If the band number is not provided (or set to `NULL`), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not `NODATA`) are considered in the test.

The distance is specified in units defined by the spatial reference system of the rasters. For this function to make sense, the source rasters must both be of the same coordinate projection, having the same `SRID`.



**Note**  
This operand will make use of any indexes that may be available on the rasters.



**Note**  
To test the spatial relationship of a raster and a geometry, use ST\_Polygon on the raster, e.g. ST\_DWithin(ST\_Polygon(raster), geometry).

Availability: 2.1.0

Examples

```
SELECT r1.rid, r2.rid, ST_DWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1 CROSS JOIN dummy_rast r2 WHERE r1.rid = 2;
```

| rid | rid | st_dwithin |
|-----|-----|------------|
| 2   | 1   | f          |
| 2   | 2   | t          |

See Also

ST\_Within, ST\_DFullyWithin

10.17.13 ST\_DFullyWithin

ST\_DFullyWithin — Return true if rasters rastA and rastB are fully within the specified distance of each other.

Synopsis

boolean **ST\_DFullyWithin**( raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance\_of\_srid );  
boolean **ST\_DFullyWithin**( raster rastA , raster rastB , double precision distance\_of\_srid );

Description

Return true if rasters rastA and rastB are fully within the specified distance of each other. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.

The distance is specified in units defined by the spatial reference system of the rasters. For this function to make sense, the source rasters must both be of the same coordinate projection, having the same SRID.



**Note**  
This operand will make use of any indexes that may be available on the rasters.



**Note**  
To test the spatial relationship of a raster and a geometry, use ST\_Polygon on the raster, e.g. ST\_DFullyWithin(ST\_Polygon(raster), geometry).

Availability: 2.1.0

## Examples

```
SELECT r1.rid, r2.rid, ST_DFullyWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1 ↵
CROSS JOIN dummy_rast r2 WHERE r1.rid = 2;
```

| rid | rid | st_dfullywithin |
|-----|-----|-----------------|
| 2   | 1   | f               |
| 2   | 2   | t               |

## See Also

[ST\\_Within](#), [ST\\_DWithin](#)

## 10.18 Raster Tips

### 10.18.1 Out-DB Rasters

#### 10.18.1.1 Directory containing many files

When GDAL opens a file, GDAL eagerly scans the directory of that file to build a catalog of other files. If this directory contains many files (e.g. thousands, millions), opening that file becomes extremely slow (especially if that file happens to be on a network drive such as NFS).

To control this behavior, GDAL provides the following environment variable: [GDAL\\_DISABLE\\_READDIR\\_ON\\_OPEN](#). Set `GDAL_DISABLE_READDIR_ON_OPEN` to `TRUE` to disable directory scanning.

In Ubuntu (and assuming you are using PostgreSQL's packages for Ubuntu), `GDAL_DISABLE_READDIR_ON_OPEN` can be set in `/etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/environment` (where `POSTGRESQL_VERSION` is the version of PostgreSQL, e.g. 9.6 and `CLUSTER_NAME` is the name of the cluster, e.g. maindb). You can also set PostGIS environment variables here as well.

```
environment variables for postmaster process
This file has the same syntax as postgresql.conf:
VARIABLE = simple_value
VARIABLE2 = 'any value!'
I. e. you need to enclose any value which does not only consist of letters,
numbers, and '-', '_', '.' in single quotes. Shell commands are not
evaluated.
POSTGIS_GDAL_ENABLED_DRIVERS = 'ENABLE_ALL'

POSTGIS_ENABLE_OUTDB_RASTERS = 1

GDAL_DISABLE_READDIR_ON_OPEN = 'TRUE'
```

#### 10.18.1.2 Maximum Number of Open Files

The maximum number of open files permitted by Linux and PostgreSQL are typically conservative (typically 1024 open files per process) given the assumption that the system is consumed by human users. For Out-DB Rasters, a single valid query can easily exceed this limit (e.g. a dataset of 10 year's worth of rasters with one raster for each day containing minimum and maximum temperatures and we want to know the absolute min and max value for a pixel in that dataset).

The easiest change to make is the following PostgreSQL setting: [max\\_files\\_per\\_process](#). The default is set to 1000, which is far too low for Out-DB Rasters. A safe starting value could be 65536 but this really depends on your datasets and the queries run against those datasets. This setting can only be made on server start and probably only in the PostgreSQL configuration file (e.g. `/etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/postgresql.conf` in Ubuntu environments).

```
...
- Kernel Resource Usage -

max_files_per_process = 65536 # min 25
 # (change requires restart)
...
```

The major change to make is the Linux kernel's open files limits. There are two parts to this:

- Maximum number of open files for the entire system
- Maximum number of open files per process

#### 10.18.1.2.1 Maximum number of open files for the entire system

You can inspect the current maximum number of open files for the entire system with the following example:

```
$ sysctl -a | grep fs.file-max
fs.file-max = 131072
```

If the value returned is not large enough, add a file to `/etc/sysctl.d/` as per the following example:

```
$ echo "fs.file-max = 6145324"
>
> /etc/sysctl.d/fs.conf

$ cat /etc/sysctl.d/fs.conf
fs.file-max = 6145324

$ sysctl -p --system
* Applying /etc/sysctl.d/fs.conf ...
fs.file-max = 2097152
* Applying /etc/sysctl.conf ...

$ sysctl -a | grep fs.file-max
fs.file-max = 6145324
```

#### 10.18.1.2.2 Maximum number of open files per process

We need to increase the maximum number of open files per process for the PostgreSQL server processes.

To see what the current PostgreSQL service processes are using for maximum number of open files, do as per the following example (make sure to have PostgreSQL running):

```
$ ps aux | grep postgres
postgres 31713 0.0 0.4 179012 17564 pts/0 S Dec26 0:03 /home/dustymugs/devel/ ↵
 postgresql/sandbox/10/usr/local/bin/postgres -D /home/dustymugs/devel/postgresql/sandbox ↵
 /10/pgdata
postgres 31716 0.0 0.8 179776 33632 ? Ss Dec26 0:01 postgres: checkpointer ↵
 process
postgres 31717 0.0 0.2 179144 9416 ? Ss Dec26 0:05 postgres: writer process
postgres 31718 0.0 0.2 179012 8708 ? Ss Dec26 0:06 postgres: wal writer ↵
 process
postgres 31719 0.0 0.1 179568 7252 ? Ss Dec26 0:03 postgres: autovacuum ↵
 launcher process
postgres 31720 0.0 0.1 34228 4124 ? Ss Dec26 0:09 postgres: stats collector ↵
 process
postgres 31721 0.0 0.1 179308 6052 ? Ss Dec26 0:00 postgres: bgworker: ↵
 logical replication launcher
```

```
$ cat /proc/31718/limits
Limit Soft Limit Hard Limit Units
Max cpu time unlimited unlimited seconds
Max file size unlimited unlimited bytes
Max data size unlimited unlimited bytes
Max stack size 8388608 unlimited bytes
Max core file size 0 unlimited bytes
Max resident set unlimited unlimited bytes
Max processes 15738 15738 processes
Max open files 1024 4096 files
Max locked memory 65536 65536 bytes
Max address space unlimited unlimited bytes
Max file locks unlimited unlimited locks
Max pending signals 15738 15738 signals
Max msgqueue size 819200 819200 bytes
Max nice priority 0 0
Max realtime priority 0 0
Max realtime timeout unlimited unlimited us
```

In the example above, we inspected the open files limit for Process 31718. It doesn't matter which PostgreSQL process, any of them will do. The response we are interested in is *Max open files*.

We want to increase *Soft Limit* and *Hard Limit* of *Max open files* to be greater than the value we specified for the PostgreSQL setting `max_files_per_process`. In our example, we set `max_files_per_process` to 65536.

In Ubuntu (and assuming you are using PostgreSQL's packages for Ubuntu), the easiest way to change the *Soft Limit* and *Hard Limit* is to edit `/etc/init.d/postgresql` (SysV) or `/lib/systemd/system/postgresql*.service` (systemd).

Let's first address the SysV Ubuntu case where we add `ulimit -H -n 262144` and `ulimit -n 131072` to `/etc/init.d/postgresql`.

```
...
case "$1" in
 start|stop|restart|reload)
 if ["$1" = "start"]; then
 create_socket_directory
 fi
 if [-z "`pg_lsclusters -h`"]; then
 log_warning_msg 'No PostgreSQL clusters exist; see "man pg_createcluster"'
 exit 0
 fi

 ulimit -H -n 262144
 ulimit -n 131072

 for v in $versions; do
 $1 $v || EXIT=$?
 done
 exit ${EXIT:-0}
 ;;
 status)
 ...
```

Now to address the systemd Ubuntu case. We will add `LimitNOFILE=131072` to every `/lib/systemd/system/postgresql*.service` file in the **[Service]** section.

```
...
[Service]

LimitNOFILE=131072

...
```

```
[Install]
WantedBy=multi-user.target
...
```

After making the necessary systemd changes, make sure to reload the daemon

```
systemctl daemon-reload
```



## Chapter 11

# Extras de PostGIS

Este capítulo documenta las características encontradas en la carpeta Extras de fuente de tarballs y fuente de repositorio de PostGIS. Estos no siempre son empaquetados con las versiones binarias de PostGIS, pero son generalmente plpgsql básicos o scripts de shell estándar que pueden ser ejecutados tal cual.

### 11.1 Normalizador de Direcciones

Esta es una bifurcación **del estandarizador PAGC** (el código original para esta parte era **el estandarizador de direcciones PAGC PostgreSQL**).

El normalizador de direcciones es un analizador de direcciones de una sola línea que toma una dirección de entrada y la normaliza basándose en un conjunto de reglas almacenadas en una tabla y en las tablas de lex y gaz

El código está construido en una única librería de extensiones postgresql llamada `address_standardizer` que puede ser instalada con `CREATE EXTENSION address_standardizer;`. A demás de la extensión `address_standardizer`, una extensión de datos de ejemplo llamada `address_standardizer_data_us` es construida, la cual contiene tablas de gaz, lex, y rules para datos de Estados Unidos. Estas extensiones se pueden instalar mediante `CREATE EXTENSION address_standardizer_data_us;`

El código para esta extensión se puede encontrar en PostGIS `extensions/address_standardizer` y es actualmente autocontenido.

Para instrucciones de instalación, consulte: Section 2.3.

#### 11.1.1 Cómo funciona el analizador

El analizador trabaja de derecha a izquierda localizando primero los macro elementos para el código postal, estado/provincia, ciudad y a continuación los micro elementos para determinar si se está tratando con un número de casa y calle o una intersección de calle o un hito. Actualmente no busca por un código o nombre de país, pero podría ser introducido en el futuro.

**Código de país** Se supone que es de EE. UU. o CA según: código postal de estado/provincia como EE. UU. o Canadá, como EE. UU. o Canadá más EE. UU. .

**Código postal** Éstos se reconocen utilizando expresiones regulares compatibles con Perl. Estos regexs están actualmente en el `parseaddress-api.c` y es relativamente simple hacer cambios si es necesario.

**Estado/provincia** Éstos son reconocidos utilizando expresiones regulares compatibles con Perl. Estos regexs están actualmente en el `parseaddress-api.c` pero podrían ser movidos e incluidos en el futuro para facilitar el mantenimiento.

## 11.1.2 Tipos de Address Standardizer

### 11.1.2.1 stdaddr

**stdaddr** — Un tipo compuesto que consiste en los elementos de una dirección. Este es el tipo de retorno para la función `standardize_address`.

#### Descripción

Un tipo compuesto que consta de elementos de una dirección. Este es el tipo de retorno de la `standardize_address` función. Algunas descripciones de los elementos se han tomado de [Atributos postales de PAGC](#).

Los números de token indican el número de referencia de salida [rules table](#)



This method needs `address_standardizer` extension.

**construyendo** Es texto (token número 0): se refiere al número o nombre del edificio. Identificadores y tipos de edificios no analizados. Generalmente en blanco para la mayoría de direcciones.

**house\_num** Es un texto (ficha número 1): este es el número de la calle en una calle. Ejemplo 75 en 75 State Street.

**predir** Es texto (token número 2): NOMBRE DE LA CALLE PRE-DIRECCIONAL como Norte, Sur, Este, Oeste, etc.

**qual** Es texto (token número 3): NOMBRE DE CALLE PRE-MODIFICADOR Ejemplo *VIEJO* en 3715 VIEJA CARRETERA 99.

**pretipo** es texto (token número 4): TIPO DE PREFIJO DE CALLE

**nombre** es texto (token número 5): NOMBRE DE LA CALLE

**suftype** es texto (token número 6): TIPO DE POSTE DE CALLE, p. ej. St, Ave, Cir. Un tipo de calle que sigue al nombre de la calle raíz. Ejemplo *STREET* en 75 State Street.

**sufdir** es texto (token número 7): STREET POST-DIRECCIONAL Un modificador direccional que sigue al nombre de la calle. Ejemplo *WEST* en 3715 TENTH AVENUE WEST.

**ruralroute** Es texto (ficha número 8): RUTA RURAL. Ejemplo 7 en RR 7.

**extra** Es texto: Información adicional como el número de piso.

**ciudad** es texto (token número 10): Ejemplo Boston.

**estado** es texto (token número 11): Ejemplo MASSACHUSETTS

**país** es texto (token número 12): Ejemplo EE. UU.

**código postal** es el texto CÓDIGO POSTAL (CÓDIGO POSTAL) (número de ficha 13): Ejemplo 02109

**box** es el texto NÚMERO DE CAJA POSTAL (token número 14 y 15): Ejemplo 02109

**unidad** is text Apartment number or Suite Number (token number 17): Example 3B in APT 3B.

## 11.1.3 Tipos de Address Standardizer

### 11.1.3.1 rules table

**rules table** — The rules table contains a set of rules that maps address input sequence tokens to standardized output sequence. A rule is defined as a set of input tokens followed by -1 (terminator) followed by set of output tokens followed by -1 followed by number denoting kind of rule followed by ranking of rule.

## Descripción

Una tabla de reglas debe tener al menos las siguientes columnas, aunque se le permite agregar más para sus propios usos.

**id** Llave primaria de la tabla

**regla** text field denoting the rule. Details at [PAGC Address Standardizer Rule records](#).

A rule consists of a set of non-negative integers representing input tokens, terminated by a -1, followed by an equal number of non-negative integers representing postal attributes, terminated by a -1, followed by an integer representing a rule type, followed by an integer representing the rank of the rule. The rules are ranked from 0 (lowest) to 17 (highest).

So for example the rule 2 0 2 22 3 -1 5 5 6 7 3 -1 2 6 maps to sequence of output tokens *TYPE NUMBER TYPE DIRECT QUALIF* to the output sequence *STREET STREET SUFTYP SUFDIR QUALIF*. The rule is an ARC\_C rule of rank 6.

Numbers for corresponding output tokens are listed in [stdaddr](#).

## Tokens de entrada

Each rule starts with a set of input tokens followed by a terminator -1. Valid input tokens excerpted from [PAGC Input Tokens](#) are as follows:

### Form-Based Input Tokens

**AMPERS** (13). El ampersand (&) se utiliza frecuentemente para abreviar la letra "y".

**DASH** (9). Un carácter de puntuación.

**DOUBLE** (21). Secuencia de dos letras. A menudo se utilizan como identificadores.

**FRACT** (25). Las fracciones a veces se usan en números cívicos o números de unidad.

**MIXED** (23). Una cadena alfanumérica que contiene letras y dígitos. Se utiliza para identificadores.

**NUMBER** (0). Una cadena de dígitos.

**ORD** (15). Representaciones como Primera o 1ra. Se utiliza a menudo en nombres de calles.

**ORD** (18). Una sola letra.

**WORD** (1). Una palabra es una cadena de letras de longitud arbitraria. Una sola letra puede ser SINGLE y una WORD.

### Function-based Input Tokens

**BOXH** (14). Palabras utilizadas para denotar casillas postales. Por ejemplo *Box* o *PO Box*.

**BUILDH** (19). Words used to denote buildings or building complexes, usually as a prefix. For example: *Tower* in *Tower 7A*.

**BUILDT** (24). Words and abbreviations used to denote buildings or building complexes, usually as a suffix. For example: *Shopping Centre*.

**DIRECTO** (22). Words used to denote directions, for example *North*.

**MILE** (20). Words used to denote milepost addresses.

**ROAD** (6). Words and abbreviations used to denote highways and roads. For example: the *Interstate* in *Interstate 5*

**RR** (8). Words and abbreviations used to denote rural routes. *RR*.

**TYPE** (2). Words and abbreviation used to denote street types. For example: *ST* or *AVE*.

**UNITH** (16). Words and abbreviation used to denote internal subaddresses. For example, *APT* or *UNIT*.

### Postal Type Input Tokens

**QUINT** (28). Un número de 5 dígitos. Identifica un código postal

**QUAD** (29). A 4 digit number. Identifies ZIP4.

**PCH** (27). A 3 character sequence of letter number letter. Identifies an FSA, the first 3 characters of a Canadian postal code.

**PCT** (26). A 3 character sequence of number letter number. Identifies an LDU, the last 3 characters of a Canadian postal code.

### Stopwords

STOPWORDS combine with WORDS. In rules a string of multiple WORDs and STOPWORDS will be represented by a single WORD token.

**STOPWORD** (7). A word with low lexical significance, that can be omitted in parsing. For example: *THE*.

### Tokens de salida

After the first -1 (terminator), follows the output tokens and their order, followed by a terminator -1. Numbers for corresponding output tokens are listed in [stdaddr](#). What are allowed is dependent on kind of rule. Output tokens valid for each rule type are listed in the section called “[Rule Types and Rank](#)”.

### Rule Types and Rank

The final part of the rule is the rule type which is denoted by one of the following, followed by a rule rank. The rules are ranked from 0 (lowest) to 17 (highest).

#### MACRO\_C

(token number = "0"). The class of rules for parsing MACRO clauses such as *PLACE STATE ZIP*

**MACRO\_C output tokens** (excerpted from <http://www.pgcgeo.org/docs/html/pagc-12.html#--r-typ-->).

**CITY** (token number "10"). Example "Albany"

**STATE** (token number "11"). Example "NY"

**NATION** (token number "12"). This attribute is not used in most reference files. Example "USA"

**POSTAL** (token number "13"). (SADS elements "ZIP CODE" , "PLUS 4" ). This attribute is used for both the US Zip and the Canadian Postal Codes.

#### MICRO\_C

(token number = "1"). The class of rules for parsing full MICRO clauses (such as House, street, sufdir, predir, pretyp, suftype, qualif) (ie ARC\_C plus CIVIC\_C). These rules are not used in the build phase.

**MICRO\_C output tokens** (excerpted from <http://www.pgcgeo.org/docs/html/pagc-12.html#--r-typ-->).

**HOUSE** Es un texto (ficha número 1): este es el número de la calle en una calle. Ejemplo 75 en 75 State Street.

**predir** Es texto (token número 2): NOMBRE DE LA CALLE PRE-DIRECCIONAL como Norte, Sur, Este, Oeste, etc.

**qual** Es texto (token número 3): NOMBRE DE CALLE PRE-MODIFICADOR Ejemplo *VIEJO* en 3715 VIEJA CARRETERA 99.

**pretipo** es texto (token número 4): TIPO DE PREFIJO DE CALLE

**street** es texto (token número 5): NOMBRE DE LA CALLE

**suftype** es texto (token número 6): TIPO DE POSTE DE CALLE, p. ej. St, Ave, Cir. Un tipo de calle que sigue al nombre de la calle raíz. Ejemplo *STREET* en 75 State Street.

**sufdir** es texto (token número 7): STREET POST-DIRECTIONAL Un modificador direccional que sigue al nombre de la calle. Ejemplo *WEST* en 3715 TENTH AVENUE WEST.

### ARC\_C

(token number = "2"). The class of rules for parsing MICRO clauses, excluding the HOUSE attribute. As such uses same set of output tokens as MICRO\_C minus the HOUSE token.

### CIVIC\_C

(token number = "3"). The class of rules for parsing the HOUSE attribute.

### EXTRA\_C

(token number = "4"). The class of rules for parsing EXTRA attributes - attributes excluded from geocoding. These rules are not used in the build phase.

**EXTRA\_C output tokens** (excerpted from <http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ-->).

**BLDNG** (token number 0): Unparsed building identifiers and types.

**BOXH** (token number 14): The **BOX** in BOX 3B

**BOXT** (token number 15): The **3B** in BOX 3B

**RR** (token number 8): The **RR** in RR 7

**UNITH** (token number 16): The **APT** in APT 3B

**UNITT** (token number 17): The **3B** in APT 3B

**UNKWN** (token number 9): An otherwise unclassified output.

#### 11.1.3.2 lex table

lex table — A lex table is used to classify alphanumeric input and associate that input with (a) input tokens ( See the section called “**Tokens de entrada**”) and (b) standardized representations.

#### Descripción

A lex (short for lexicon) table is used to classify alphanumeric input and associate that input with the section called “**Tokens de entrada**” and (b) standardized representations. Things you will find in these tables are ONE mapped to stdword: 1.

A lex has at least the following columns in the table. You may add

**id** Llave primaria de la tabla

**seq** integer: definition number?

**word** text: the input word

**stdword** text: the standardized replacement word

**token** integer: the kind of word it is. Only if it is used in this context will it be replaced. Refer to **PAGC Tokens**.

#### 11.1.3.3 gaz table

gaz table — A gaz table is used to standardize place names and associate that input with (a) input tokens ( See the section called “**Tokens de entrada**”) and (b) standardized representations.

## Descripción

A gaz (short for gazeteer) table is used to standardize place names and associate that input with the section called “**Tokens de entrada**” and (b) standardized representations. For example if you are in US, you may load these with State Names and associated abbreviations.

A gaz table has at least the following columns in the table. You may add more columns if you wish for your own purposes.

**id** Llave primaria de la tabla

**seq** integer: definition number? - identifier used for that instance of the word

**word** text: the input word

**stdword** text: the standardized replacement word

**token** integer: the kind of word it is. Only if it is used in this context will it be replaced. Refer to **PAGC Tokens**.

## 11.1.4 Funciones de Address Standardizer

### 11.1.4.1 debug\_standardize\_address

`debug_standardize_address` — Returns a json formatted text listing the parse tokens and standardizations

## Synopsis

text **debug\_standardize\_address**(text lextab, text gaztab, text rultab, text micro, text macro=NULL);

## Descripción

This is a function for debugging address standardizer rules and lex/gaz mappings. It returns a json formatted text that includes the matching rules, mapping of tokens, and best standardized address **stdaddr** form of an input address utilizing **lex table** table name, **gaz table**, and **rules table** table names and an address.

For single line addresses use just `micro`

For two line address A `micro` consisting of standard first line of postal address e.g. `house_num street`, and a macro consisting of standard postal second line of an address e.g `city, state postal_code country`.

Elements returned in the json document are

**input\_tokens** For each word in the input address, returns the position of the word, token categorization of the word, and the standard word it is mapped to. Note that for some input words, you might get back multiple records because some inputs can be categorized as more than one thing.

**rules** The set of rules matching the input and the corresponding score for each. The first rule (highest scoring) is what is used for standardization

**stdaddr** The standardized address elements **stdaddr** that would be returned when running **standardize\_address**

Availability: 3.4.0



This method needs `address_standardizer` extension.

Ejemplos

Using address\_standardizer\_data\_us extension

```
CREATE EXTENSION address_standardizer_data_us; -- only needs to be done once
```

Variant 1: Single line address and returning the input tokens

```
SELECT it->
>'pos' AS position, it->
>'word' AS word, it->
>'stdword' AS standardized_word,
 it->
>'token' AS token, it->
>'token-code' AS token_code
 FROM jsonb(
 debug_standardize_address('us_lex',
 'us_gaz', 'us_rules', 'One Devonshire Place, PH 301, Boston, MA 02109')
) AS s, jsonb_array_elements(s->'input_tokens') AS it;
```

| position | word       | standardized_word | token  | token_code |
|----------|------------|-------------------|--------|------------|
| 0        | ONE        | 1                 | NUMBER | 0          |
| 0        | ONE        | 1                 | WORD   | 1          |
| 1        | DEVONSHIRE | DEVONSHIRE        | WORD   | 1          |
| 2        | PLACE      | PLACE             | TYPE   | 2          |
| 3        | PH         | PATH              | TYPE   | 2          |
| 3        | PH         | PENTHOUSE         | UNITT  | 17         |
| 4        | 301        | 301               | NUMBER | 0          |

(7 rows)

Variant 2: Multi line address and returning first rule input mappings and score

```
SELECT (s->'rules'->0->
>'score')::numeric AS score, it->
>'pos' AS position,
 it->
>'input-word' AS word, it->
>'input-token' AS input_token, it->
>'mapped-word' AS standardized_word,
 it->
>'output-token' AS output_token
 FROM jsonb(
 debug_standardize_address('us_lex',
 'us_gaz', 'us_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109')
) AS s, jsonb_array_elements(s->'rules'->0->'rule_tokens') AS it;
```

| score    | position | word       | input_token | standardized_word | output_token |
|----------|----------|------------|-------------|-------------------|--------------|
| 0.876250 | 0        | ONE        | NUMBER      | 1                 | HOUSE        |
| 0.876250 | 1        | DEVONSHIRE | WORD        | DEVONSHIRE        | STREET       |
| 0.876250 | 2        | PLACE      | TYPE        | PLACE             | SUFTYP       |
| 0.876250 | 3        | PH         | UNITT       | PENTHOUSE         | UNITT        |
| 0.876250 | 4        | 301        | NUMBER      | 301               | UNITT        |

(5 rows)

Ver también

[stdaddr](#), [rules table](#), [lex table](#), [gaz table](#), [Pgac\\_Normalize\\_Address](#)

#### 11.1.4.2 parse\_address

`parse_address` — Takes a 1 line address and breaks into parts

## Synopsis

```
record parse_address(text address);
```

### Descripción

Returns takes an address as input, and returns a record output consisting of fields *num*, *street*, *street2*, *address1*, *city*, *state*, *zip*, *zipplus*, *country*.

Disponibilidad: 2.2.0



This method needs `address_standardizer` extension.

## Ejemplos

## Single Addresss

```
SELECT num, street, city, zip, zipplus
 FROM parse_address('1 Devonshire Place, Boston, MA 02109-1234') AS a;
```

| num | street           | city   | zip   | zipplus |
|-----|------------------|--------|-------|---------|
| 1   | Devonshire Place | Boston | 02109 | 1234    |

## Table of addresses

```
-- basic table
CREATE TABLE places(addid serial PRIMARY KEY, address text);

INSERT INTO places(address)
VALUES ('529 Main Street, Boston MA, 02129'),
 ('77 Massachusetts Avenue, Cambridge, MA 02139'),
 ('25 Wizard of Oz, Walaford, KS 99912323'),
 ('26 Capen Street, Medford, MA'),
 ('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
 ('950 Main Street, Worcester, MA 01610');

-- parse the addresses
-- if you want all fields you can use (a).*
SELECT addid, (a).num, (a).street, (a).city, (a).state, (a).zip, (a).zipplus
FROM (SELECT addid, parse_address(address) As a
 FROM places) AS p;
```

| addid | num | street               | city      | state | zip   | zipplus |
|-------|-----|----------------------|-----------|-------|-------|---------|
| 1     | 529 | Main Street          | Boston    | MA    | 02129 |         |
| 2     | 77  | Massachusetts Avenue | Cambridge | MA    | 02139 |         |
| 3     | 25  | Wizard of Oz         | Walaford  | KS    | 99912 | 323     |
| 4     | 26  | Capen Street         | Medford   | MA    |       |         |
| 5     | 124 | Mount Auburn St      | Cambridge | MA    | 02138 |         |
| 6     | 950 | Main Street          | Worcester | MA    | 01610 |         |



**Ver también****11.1.4.3 standardize\_address**

`standardize_address` — Returns an `stdaddr` form of an input address utilizing `lex`, `gaz`, and `rule` tables.

**Synopsis**

```
stdaddr standardize_address(text lextab, text gaztab, text rultab, text address);
stdaddr standardize_address(text lextab, text gaztab, text rultab, text micro, text macro);
```

**Descripción**

Returns an `stdaddr` form of an input address utilizing `lex table` table name, `gaz table`, and `rules table` table names and an address.

Variant 1: Takes an address as a single line.

Variant 2: Takes an address as 2 parts. A `micro` consisting of standard first line of postal address e.g. `house_num street`, and a `macro` consisting of standard postal second line of an address e.g. `city, state postal_code country`.

Disponibilidad: 2.2.0



This method needs `address_standardizer` extension.

**Ejemplos**

Using `address_standardizer_data_us` extension

```
CREATE EXTENSION address_standardizer_data_us; -- only needs to be done once
```

Variant 1: Single line address. This doesn't work well with non-US addresses

```
SELECT house_num, name, suftype, city, country, state, unit FROM standardize_address(' ←
 us_lex',
 'us_gaz', 'us_rules', 'One Devonshire Place, PH 301, Boston, MA ←
 02109');
```

| house_num | name       | suftype | city   | country | state         | unit            |
|-----------|------------|---------|--------|---------|---------------|-----------------|
| 1         | DEVONSHIRE | PLACE   | BOSTON | USA     | MASSACHUSETTS | # PENTHOUSE 301 |

Using tables packaged with `tiger geocoder`. This example only works if you installed `postgis_tiger_geocoder`.

```
SELECT * FROM standardize_address('tiger.pagc_lex',
 'tiger.pagc_gaz', 'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA ←
 02109-1234');
```

Make easier to read we'll dump output using `hstore` extension `CREATE EXTENSION hstore`; you need to install

```
SELECT (each(hstore(p))).*
FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
 'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA 02109') As p;
```

| key  | value      |
|------|------------|
| box  |            |
| city | BOSTON     |
| name | DEVONSHIRE |
| qual |            |

```

unit | # PENTHOUSE 301
extra |
state | MA
predir |
sufdir |
country | USA
pretype |
suftype | PL
building |
postcode | 02109
house_num | 1
ruralroute |
(16 rows)

```

### Variant 2: As a two part Address

```

SELECT (each(hstore(p))).*
FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
 'tiger.pagc_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109, US') As p;

```

| key        | value           |
|------------|-----------------|
| box        |                 |
| city       | BOSTON          |
| name       | DEVONSHIRE      |
| qual       |                 |
| unit       | # PENTHOUSE 301 |
| extra      |                 |
| state      | MA              |
| predir     |                 |
| sufdir     |                 |
| country    | USA             |
| pretype    |                 |
| suftype    | PL              |
| building   |                 |
| postcode   | 02109           |
| house_num  | 1               |
| ruralroute |                 |

(16 rows)

### Ver también

[stdaddr](#), [rules table](#), [lex table](#), [gaz table](#), [Pagc\\_Normalize\\_Address](#)

## 11.2 Geocodificador Tiger

Hay un par de geocodificadores de software libre para PostGIS, que a diferencia del geocodificador tiger tienen la ventaja de soporte para geocodificación multi-país.

- **Nominatim** uses OpenStreetMap gazeteer formatted data. It requires osm2pgsql for loading the data, PostgreSQL 8.4+ and PostGIS 1.5+ to function. It is packaged as a webservice interface and seems designed to be called as a webservice. Just like the tiger geocoder, it has both a geocoder and a reverse geocoder component. From the documentation, it is unclear if it has a pure SQL interface like the tiger geocoder, or if a good deal of the logic is implemented in the web interface.
- **GIS Graphy** también utiliza PostGIS y como Nominatim trabaja con datos de OpenStreetMap (OSM). Viene con un cargador para cargar datos OSM y, al igual que Nominatim es capaz de geocodificar no solo USA. Similar a Nominatim, se ejecuta como servicio web y se apoya en Java 1.5, aplicaciones Servlet, Soir. GisGraphy es multiplataforma y también tiene un geocodificador inverso entre otras buenas características.

### 11.2.1 Drop\_Indexes\_Generate\_Script

**Drop\_Indexes\_Generate\_Script** — Generates a script that drops all non-primary key and non-unique indexes on tiger schema and user specified schema. Defaults schema to `tiger_data` if no schema is specified.

#### Synopsis

text **Drop\_Indexes\_Generate\_Script**(text param\_schema=tiger\_data);

#### Descripción

Generates a script that drops all non-primary key and non-unique indexes on tiger schema and user specified schema. Defaults schema to `tiger_data` if no schema is specified.

This is useful for minimizing index bloat that may confuse the query planner or take up unnecessary space. Use in combination with [Install\\_Missing\\_Indexes](#) to add just the indexes used by the geocoder.

Disponibilidad: 2.0.0

#### Ejemplos

```
SELECT drop_indexes_generate_script() As actionsql;
actionsql

DROP INDEX tiger.idx_tiger_countysub_lookup_lower_name;
DROP INDEX tiger.idx_tiger_edges_countyfp;
DROP INDEX tiger.idx_tiger_faces_countyfp;
DROP INDEX tiger.tiger_place_the_geom_gist;
DROP INDEX tiger.tiger_edges_the_geom_gist;
DROP INDEX tiger.tiger_state_the_geom_gist;
DROP INDEX tiger.idx_tiger_addr_least_address;
DROP INDEX tiger.idx_tiger_addr_tlid;
DROP INDEX tiger.idx_tiger_addr_zip;
DROP INDEX tiger.idx_tiger_county_countyfp;
DROP INDEX tiger.idx_tiger_county_lookup_lower_name;
DROP INDEX tiger.idx_tiger_county_lookup_snd_name;
DROP INDEX tiger.idx_tiger_county_lower_name;
DROP INDEX tiger.idx_tiger_county_snd_name;
DROP INDEX tiger.idx_tiger_county_the_geom_gist;
DROP INDEX tiger.idx_tiger_countysub_lookup_snd_name;
DROP INDEX tiger.idx_tiger_cousub_countyfp;
DROP INDEX tiger.idx_tiger_cousub_cousubfp;
DROP INDEX tiger.idx_tiger_cousub_lower_name;
DROP INDEX tiger.idx_tiger_cousub_snd_name;
DROP INDEX tiger.idx_tiger_cousub_the_geom_gist;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_least_address;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_tlid;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_zip;
DROP INDEX tiger_data.idx_tiger_data_ma_county_countyfp;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_snd_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_snd_name;
:
:
```

#### Ver también

[Install\\_Missing\\_Indexes](#), [Missing\\_Indexes\\_Generate\\_Script](#)

### 11.2.2 Drop\_Nation\_Tables\_Generate\_Script

**Drop\_Nation\_Tables\_Generate\_Script** — Generates a script that drops all tables in the specified schema that start with `county_all`, `state_all` or state code followed by `county` or `state`.

#### Synopsis

`text Drop_Nation_Tables_Generate_Script(text param_schema=tiger_data);`

#### Descripción

Generates a script that drops all tables in the specified schema that start with `county_all`, `state_all` or state code followed by `county` or `state`. This is needed if you are upgrading from `tiger_2010` to `tiger_2011` data.

Disponibilidad: 2.1.0

#### Ejemplos

```
SELECT drop_nation_tables_generate_script();
DROP TABLE tiger_data.county_all;
DROP TABLE tiger_data.county_all_lookup;
DROP TABLE tiger_data.state_all;
DROP TABLE tiger_data.ma_county;
DROP TABLE tiger_data.ma_state;
```

#### Ver también

[Loader\\_Generate\\_Nation\\_Script](#)

### 11.2.3 Drop\_State\_Tables\_Generate\_Script

**Drop\_State\_Tables\_Generate\_Script** — Generates a script that drops all tables in the specified schema that are prefixed with the state abbreviation. Defaults schema to `tiger_data` if no schema is specified.

#### Synopsis

`text Drop_State_Tables_Generate_Script(text param_state, text param_schema=tiger_data);`

#### Descripción

Generates a script that drops all tables in the specified schema that are prefixed with the state abbreviation. Defaults schema to `tiger_data` if no schema is specified. This function is useful for dropping tables of a state just before you reload a state in case something went wrong during your previous load.

Disponibilidad: 2.0.0

## Ejemplos

```
SELECT drop_state_tables_generate_script('PA');
DROP TABLE tiger_data.pa_addr;
DROP TABLE tiger_data.pa_county;
DROP TABLE tiger_data.pa_county_lookup;
DROP TABLE tiger_data.pa_cousub;
DROP TABLE tiger_data.pa_edges;
DROP TABLE tiger_data.pa_faces;
DROP TABLE tiger_data.pa_featnames;
DROP TABLE tiger_data.pa_place;
DROP TABLE tiger_data.pa_state;
DROP TABLE tiger_data.pa_zip_lookup_base;
DROP TABLE tiger_data.pa_zip_state;
DROP TABLE tiger_data.pa_zip_state_loc;
```

## Ver también

[Loader\\_Generate\\_Script](#)

### 11.2.4 Geocode

Geocode — Takes in an address as a string (or other normalized address) and outputs a set of possible locations which include a point geometry in NAD 83 long lat, a normalized address for each, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Can optionally pass in maximum results, defaults to 10, and restrict\_region (defaults to NULL)

## Synopsis

setof record **geocode**(varchar address, integer max\_results=10, geometry restrict\_region=NULL, norm\_addy OUT addy, geometry OUT geomout, integer OUT rating);  
 setof record **geocode**(norm\_addy in\_addy, integer max\_results=10, geometry restrict\_region=NULL, norm\_addy OUT addy, geometry OUT geomout, integer OUT rating);

## Descripción

Takes in an address as a string (or already normalized address) and outputs a set of possible locations which include a point geometry in NAD 83 long lat, a `normalized_address` (addy) for each, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Uses Tiger data (edges,faces,addr), PostgreSQL fuzzy string matching (soundex,levenshtein) and PostGIS line interpolation functions to interpolate address along the Tiger edges. The higher the rating the less likely the geocode is right. The geocoded point is defaulted to offset 10 meters from center-line off to side (L/R) of street address is located on.

Enhanced: 2.0.0 to support Tiger 2010 structured data and revised some logic to improve speed, accuracy of geocoding, and to offset point from centerline to side of street address is located on. The new parameter `max_results` useful for specifying number of best results or just returning the best result.

## Examples: Basic

The below examples timings are on a 3.0 GHZ single processor Windows 7 machine with 2GB ram running PostgreSQL 9.1rc1/PostGIS 2.0 loaded with all of MA,MN,CA, RI state Tiger data loaded.

Exact matches are faster to compute (61ms)

```
SELECT g.rating, ST_X(g.geomout) As lon, ST_Y(g.geomout) As lat,
 (addy).address As stno, (addy).streetname As street,
 (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st, (←
 addy).zip
FROM geocode('75 State Street, Boston MA 02109', 1) As g;
rating | lon | lat | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----+-----
 0 | -71.0557505845646 | 42.35897920691 | 75 | State | St | Boston | MA | 02109
```

Even if zip is not passed in the geocoder can guess (took about 122-150 ms)

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktnlonlat,
 (addy).address As stno, (addy).streetname As street,
 (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st, (←
 addy).zip
FROM geocode('226 Hanover Street, Boston, MA',1) As g;
rating | wktnlonlat | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----
 1 | POINT(-71.05528 42.36316) | 226 | Hanover | St | Boston | MA | 02113
```

Can handle misspellings and provides more than one possible solution with ratings and takes longer (500ms).

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktnlonlat,
 (addy).address As stno, (addy).streetname As street,
 (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st, (←
 addy).zip
FROM geocode('31 - 37 Stewart Street, Boston, MA 02116',1) As g;
rating | wktnlonlat | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----
 70 | POINT(-71.06466 42.35114) | 31 | Stuart | St | Boston | MA | 02116
```

Using to do a batch geocode of addresses. Easiest is to set max\_results=1. Only process those not yet geocoded (have no rating).

```
CREATE TABLE addresses_to_geocode(addid serial PRIMARY KEY, address text,
 lon numeric, lat numeric, new_address text, rating integer);

INSERT INTO addresses_to_geocode(address)
VALUES ('529 Main Street, Boston MA, 02129'),
 ('77 Massachusetts Avenue, Cambridge, MA 02139'),
 ('25 Wizard of Oz, Walaford, KS 99912323'),
 ('26 Capen Street, Medford, MA'),
 ('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
 ('950 Main Street, Worcester, MA 01610');

-- only update the first 3 addresses (323-704 ms - there are caching and shared memory ←
-- effects so first geocode you do is always slower) --
-- for large numbers of addresses you don't want to update all at once
-- since the whole geocode must commit at once
-- For this example we rejoin with LEFT JOIN
-- and set to rating to -1 rating if no match
-- to ensure we don't regeocode a bad address
UPDATE addresses_to_geocode
SET (rating, new_address, lon, lat)
= (COALESCE(g.rating, -1), pprint_addy(g.addy),
 ST_X(g.geomout)::numeric(8,5), ST_Y(g.geomout)::numeric(8,5))
FROM (SELECT addid, address
 FROM addresses_to_geocode
 WHERE rating IS NULL ORDER BY addid LIMIT 3) As a
LEFT JOIN LATERAL geocode(a.address,1) As g ON true
WHERE a.addid = addresses_to_geocode.addid;
```

```
result

Query returned successfully: 3 rows affected, 480 ms execution time.

SELECT * FROM addresses_to_geocode WHERE rating is not null;

addid | address | lon | lat | ↵
-----+-----+-----+-----+-----
1 | 529 Main Street, Boston MA, 02129 | -71.07177 | 42.38357 | 529 Main St, ↵
 | Boston, MA 02129 | 0
2 | 77 Massachusetts Avenue, Cambridge, MA 02139 | -71.09396 | 42.35961 | 77 ↵
 | Massachusetts Ave, Cambridge, MA 02139 | 0
3 | 25 Wizard of Oz, Walaford, KS 99912323 | -97.92913 | 38.12717 | Willowbrook, ↵
 | KS 67502 | 108
(3 rows)
```

Examples: Using Geometry filter

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
 (addy).address As stno, (addy).streetname As street,
 (addy).streettypeabbrev As styp,
 (addy).location As city, (addy).stateabbrev As st, (addy).zip
FROM geocode('100 Federal Street, MA',
 3,
 (SELECT ST_Union(the_geom)
 FROM place WHERE statefp = '25' AND name = 'Lynn')::geometry
) As g;

rating | wktlonlat | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----
7 | POINT(-70.96796 42.4659) | 100 | Federal | St | Lynn | MA | 01905
16 | POINT(-70.96786 42.46853) | NULL | Federal | St | Lynn | MA | 01905
(2 rows)

Time: 622.939 ms
```

Ver también

[Normalize\\_Address](#), [Pprint\\_Addy](#), [ST\\_AsText](#), [ST\\_SnapToGrid](#), [ST\\_X](#), [ST\\_Y](#)

11.2.5 Geocode\_Intersection

Geocode\_Intersection — Takes in 2 streets that intersect and a state, city, zip, and outputs a set of possible locations on the first cross street that is at the intersection, also includes a geomout as the point location in NAD 83 long lat, a normalized\_address (addy) for each location, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Can optionally pass in maximum results, defaults to 10. Uses Tiger data (edges, faces, addr), PostgreSQL fuzzy string matching (soundex, levenshtein).

Synopsis

setof record **geocode\_intersection**(text roadway1, text roadway2, text in\_state, text in\_city, text in\_zip, integer max\_results=10, norm\_addy OUT addy, geometry OUT geomout, integer OUT rating);

## Descripción

Takes in 2 streets that intersect and a state, city, zip, and outputs a set of possible locations on the first cross street that is at the intersection, also includes a point geometry in NAD 83 long lat, a normalized address for each location, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Can optionally pass in maximum results, defaults to 10. Returns `normalized_address` (addy) for each, `geomout` as the point location in nad 83 long lat, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Uses Tiger data (edges,faces,addr), PostgreSQL fuzzy string matching (soundex,levenshtein)

Disponibilidad: 2.0.0

## Examples: Basic

The below examples timings are on a 3.0 GHZ single processor Windows 7 machine with 2GB ram running PostgreSQL 9.0/PostGIS 1.5 loaded with all of MA state Tiger data loaded. Currently a bit slow (3000 ms)

Testing on Windows 2003 64-bit 8GB on PostGIS 2.0 PostgreSQL 64-bit Tiger 2011 data loaded -- (41ms)

```
SELECT pprint_addy(addy), st_astext(geomout),rating
 FROM geocode_intersection('Haverford St','Germania St', 'MA', 'Boston', ←
 '02130',1);
```

| pprint_addy                       | st_astext                  | rating |
|-----------------------------------|----------------------------|--------|
| 98 Haverford St, Boston, MA 02130 | POINT(-71.101375 42.31376) | 0      |

Even if zip is not passed in the geocoder can guess (took about 3500 ms on the windows 7 box), on the windows 2003 64-bit 741 ms

```
SELECT pprint_addy(addy), st_astext(geomout),rating
 FROM geocode_intersection('Weld', 'School', 'MA', 'Boston');
```

| pprint_addy                   | st_astext                | rating |
|-------------------------------|--------------------------|--------|
| 98 Weld Ave, Boston, MA 02119 | POINT(-71.099 42.314234) | 3      |
| 99 Weld Ave, Boston, MA 02119 | POINT(-71.099 42.314234) | 3      |

## Ver también

[Geocode](#), [Pprint\\_Addy](#), [ST\\_AsText](#)

## 11.2.6 Get\_Geocode\_Setting

`Get_Geocode_Setting` — Returns value of specific setting stored in `tiger.geocode_settings` table.

## Synopsis

text `Get_Geocode_Setting`(text setting\_name);

## Descripción

Returns value of specific setting stored in `tiger.geocode_settings` table. Settings allow you to toggle debugging of functions. Later plans will be to control rating with settings. Current list of settings are as follows:

| name                                | setting | unit | category | ↔ | short_desc |
|-------------------------------------|---------|------|----------|---|------------|
| -----+-----+-----+-----+-----+----- |         |      |          |   |            |



|                                                                                        |       |         |           |                           |   |
|----------------------------------------------------------------------------------------|-------|---------|-----------|---------------------------|---|
| debug_geocode_address                                                                  | false | boolean | debug     | outputs debug information | ↵ |
| in notice log such as queries when geocode_address is called if true                   |       |         |           |                           |   |
| debug_geocode_intersection                                                             | false | boolean | debug     | outputs debug information | ↵ |
| in notice log such as queries when geocode_intersection is called if true              |       |         |           |                           |   |
| debug_normalize_address                                                                | false | boolean | debug     | outputs debug information | ↵ |
| in notice log such as queries and intermediate expressions when normalize_address is ↵ |       |         |           |                           |   |
| called if true                                                                         |       |         |           |                           |   |
| debug_reverse_geocode                                                                  | false | boolean | debug     | if true, outputs debug    | ↵ |
| information in notice log such as queries and intermediate expressions when ↵          |       |         |           |                           |   |
| reverse_geocode                                                                        |       |         |           |                           |   |
| reverse_geocode_numbered_roads                                                         | 0     | integer | rating    | For state and county      | ↵ |
| highways, 0 - no preference in name,                                                   |       |         |           |                           |   |
| 1 - prefer the numbered ↵                                                              |       |         |           |                           |   |
| highway name, 2 - ↵                                                                    |       |         |           |                           |   |
| prefer local state/ ↵                                                                  |       |         |           |                           |   |
| county name                                                                            |       |         |           |                           |   |
| use_pagc_address_parser                                                                | false | boolean | normalize | If set to true, will try  | ↵ |
| to use the address_standardizer extension (via pagc_normalize_address)                 |       |         |           |                           |   |
| instead of tiger ↵                                                                     |       |         |           |                           |   |
| normalize_address built ↵                                                              |       |         |           |                           |   |
| one                                                                                    |       |         |           |                           |   |

Changed: 2.2.0 : default settings are now kept in a table called `geocode_settings_default`. Use customized settingsa are in `geocode_settings` and only contain those that have been set by user.

Disponibilidad: 2.1.0

### Example return debugging setting

```
SELECT get_geocode_setting('debug_geocode_address') As result;
result

false
```

### Ver también

[Set\\_Geocode\\_Setting](#)

## 11.2.7 Get\_Tract

**Get\_Tract** — Returns census tract or field from tract table of where the geometry is located. Default to returning short name of tract.

### Synopsis

text **get\_tract**(geometry loc\_geom, text output\_field=name);

### Descripción

Given a geometry will return the census tract location of that geometry. NAD 83 long lat is assumed if no spatial ref sys is specified.

**Note**

This function uses the census `tract` which is not loaded by default. If you have already loaded your state table, you can load `tract` as well as `bg`, and `tabblock` using the [Loader\\_Generate\\_Census\\_Script](#) script.

If you have not loaded your state data yet and want these additional tables loaded, do the following

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

then they will be included by the [Loader\\_Generate\\_Script](#).

Disponibilidad: 2.0.0

**Examples: Basic**

```
SELECT get_tract(ST_Point(-71.101375, 42.31376)) As tract_name;
tract_name

1203.01
```

```
--this one returns the tiger geoid
SELECT get_tract(ST_Point(-71.101375, 42.31376), 'tract_id') As tract_id;
tract_id

25025120301
```

**Ver también**

[Geocode](#) >

**11.2.8 Install\_Missing\_Indexes**

`Install_Missing_Indexes` — Finds all tables with key columns used in geocoder joins and filter conditions that are missing used indexes on those columns and will add them.

**Synopsis**

boolean `Install_Missing_Indexes()`;

**Descripción**

Finds all tables in `tiger` and `tiger_data` schemas with key columns used in geocoder joins and filters that are missing indexes on those columns and will output the SQL DDL to define the index for those tables and then execute the generated script. This is a helper function that adds new indexes needed to make queries faster that may have been missing during the load process. This function is a companion to [Missing\\_Indexes\\_Generate\\_Script](#) that in addition to generating the create index script, also executes it. It is called as part of the `update_geocode.sql` upgrade script.

Disponibilidad: 2.0.0

**Ejemplos**

```
SELECT install_missing_indexes();
install_missing_indexes

t
```

**Ver también**

[Loader\\_Generate\\_Script](#), [Missing\\_Indexes\\_Generate\\_Script](#)

**11.2.9 Loader\_Generate\_Census\_Script**

**Loader\_Generate\_Census\_Script** — Generates a shell script for the specified platform for the specified states that will download Tiger census state tract, bg, and tabblocks data tables, stage and load into `tiger_data` schema. Each state script is returned as a separate record.

**Synopsis**

```
setof text loader_generate_census_script(text[] param_states, text os);
```

**Descripción**

Generates a shell script for the specified platform for the specified states that will download Tiger data census state tract, block groups bg, and tabblocks data tables, stage and load into `tiger_data` schema. Each state script is returned as a separate record.

It uses unzip on Linux (7-zip on Windows by default) and wget to do the downloading. It uses Section 4.7.2 to load in the data. Note the smallest unit it does is a whole state. It will only process the files in the staging and temp folders.

It uses the following control tables to control the process and different OS shell syntax variations.

1. `loader_variables` keeps track of various variables such as census site, year, data and staging schemas
2. `loader_platform` profiles of various platforms and where the various executables are located. Comes with windows and linux. More can be added.
3. `loader_lookuptables` each record defines a kind of table (state, county), whether to process records in it and how to load them in. Defines the steps to import data, stage data, add, removes columns, indexes, and constraints for each. Each table is prefixed with the state and inherits from a table in the tiger schema. e.g. creates `tiger_data.ma_faces` which inherits from `tiger.faces`

Disponibilidad: 2.0.0

**Note**

[Loader\\_Generate\\_Script](#) includes this logic, but if you installed tiger geocoder prior to PostGIS 2.0.0 alpha5, you'll need to run this on the states you have already done to get these additional tables.

**Ejemplos**

Generate script to load up data for select states in Windows shell script format.

```
SELECT loader_generate_census_script (ARRAY['MA'], 'windows');
-- result --
set STATEDIR="\gisdata\www2.census.gov\geo\pvs\tiger2010st\25_Massachusetts"
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\projects\pg\pg91win\bin\
set PGPORT=5432
set PGHOST=localhost
set PGUSER=postgres
```

```

set PGPASSWORD=yourpasswordhere
set PGDATABASE=tiger_postgis20
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata

%WGETTOOL% http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent -- ←
 relative --accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
del %TMPDIR%*.* /Q
%PSQL% -c "DROP SCHEMA tiger_staging CASCADE;"
%PSQL% -c "CREATE SCHEMA tiger_staging;"
cd %STATEDIR%
for /r %%z in (*.zip) do %UNZIPTOOL% e %%z -o%TMPDIR%
cd %TMPDIR%
%PSQL% -c "CREATE TABLE tiger_data.MA_tract(CONSTRAINT pk_MA_tract PRIMARY KEY (tract_id)) ←
 INHERITS(tiger.tract); "
%SHP2PGSQL% -c -s 4269 -g the_geom -W "latin1" tl_2010_25_tract10.dbf tiger_staging. ←
 ma_tract10 | %PSQL%
%PSQL% -c "ALTER TABLE tiger_staging.MA_tract10 RENAME geoid10 TO tract_id; SELECT ←
 loader_load_staged_data(lower('MA_tract10'), lower('MA_tract')); "
%PSQL% -c "CREATE INDEX tiger_data_MA_tract_the_geom_gist ON tiger_data.MA_tract USING gist ←
 (the_geom);"
%PSQL% -c "VACUUM ANALYZE tiger_data.MA_tract;"
%PSQL% -c "ALTER TABLE tiger_data.MA_tract ADD CONSTRAINT chk_statefp CHECK (statefp = ←
 '25');"
:

```

### Generate sh script

```

STATEDIR="/gisdata/www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts"
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
WGETTOOL="/usr/bin/wget"
export PGBIN=/usr/pgsql-9.0/bin
export PGPORT=5432
export PGHOST=localhost
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata

wget http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent --relative ←
 --accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
rm -f ${TMPDIR}/*. *
${PSQL} -c "DROP SCHEMA tiger_staging CASCADE;"
${PSQL} -c "CREATE SCHEMA tiger_staging;"
cd $STATEDIR
for z in *.zip; do $UNZIPTOOL -o -d $TMPDIR $z; done
:
:

```

### Ver también

[Loader\\_Generate\\_Script](#)

### 11.2.10 Loader\_Generate\_Script

**Loader\_Generate\_Script** — Generates a shell script for the specified platform for the specified states that will download Tiger data, stage and load into `tiger_data` schema. Each state script is returned as a separate record. Latest version supports Tiger 2010 structural changes and also loads census tract, block groups, and blocks tables.

#### Synopsis

setof text **loader\_generate\_script**(text[] param\_states, text os);

#### Descripción

Generates a shell script for the specified platform for the specified states that will download Tiger data, stage and load into `tiger_data` schema. Each state script is returned as a separate record.

It uses unzip on Linux (7-zip on Windows by default) and wget to do the downloading. It uses Section 4.7.2 to load in the data. Note the smallest unit it does is a whole state, but you can overwrite this by downloading the files yourself. It will only process the files in the staging and temp folders.

It uses the following control tables to control the process and different OS shell syntax variations.

1. `loader_variables` keeps track of various variables such as census site, year, data and staging schemas
2. `loader_platform` profiles of various platforms and where the various executables are located. Comes with windows and linux. More can be added.
3. `loader_lookuptables` each record defines a kind of table (state, county), whether to process records in it and how to load them in. Defines the steps to import data, stage data, add, removes columns, indexes, and constraints for each. Each table is prefixed with the state and inherits from a table in the tiger schema. e.g. creates `tiger_data.ma_faces` which inherits from `tiger.faces`

Availability: 2.0.0 to support Tiger 2010 structured data and load census tract (tract), block groups (bg), and blocks (tabblocks) tables .



#### Note

If you are using pgAdmin 3, be warned that by default pgAdmin 3 truncates long text. To fix, change *File -> Options -> Query Tool -> Query Editor -> Max. characters per column* to larger than 50000 characters.

#### Ejemplos

Using `psql` where `gistest` is your database and `/gisdata/data_load.sh` is the file to create with the shell commands to run.

```
psql -U postgres -h localhost -d gistest -A -t \
-c "SELECT Loader_Generate_Script (ARRAY['MA'], 'gistest') "
> /gisdata/data_load.sh;
```

Generate script to load up data for 2 states in Windows shell script format.

```
SELECT loader_generate_script (ARRAY['MA','RI'], 'windows') AS result;
-- result --
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\Program Files\PostgreSQL\9.4\bin\
set PGPORT=5432
set PGHOST=localhost
```

```

set PGUSER=postgres
set PGPASSWORD=yourpasswordhere
set PGDATABASE=geocoder
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata

cd \gisdata
%WGETTOOL% ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl_*_25_* --no-parent --relative ←
--recursive --level=2 --accept=zip --mirror --reject=html
cd \gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE
:
:
```

### Generate sh script

```

SELECT loader_generate_script(ARRAY['MA','RI'], 'sh') AS result;
-- result --
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
WGETTOOL="/usr/bin/wget"
export PGBIN=/usr/lib/postgresql/9.4/bin
-- variables used by psql: https://www.postgresql.org/docs/current/static/libpq-envvars.html
export PGPORT=5432
export PGHOST=localhost
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata

cd /gisdata
wget ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl_*_25_* --no-parent --relative -- ←
recursive --level=2 --accept=zip --mirror --reject=html
cd /gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE
rm -f ${TMPDIR}/*. *
:
:
```

### Ver también

Section [2.4.1, Missing\\_Indexes\\_Generate\\_Script](#)

## 11.2.11 Loader\_Generate\_Nation\_Script

**Loader\_Generate\_Nation\_Script** — Generates a shell script for the specified platform that loads in the county and state lookup tables.

### Synopsis

```
text loader_generate_nation_script(text os);
```

### Descripción

Generates a shell script for the specified platform that loads in the `county_all`, `county_all_lookup`, `state_all` tables into `tiger_data` schema. These inherit respectively from the `county`, `county_lookup`, `state` tables in `tiger` schema.

It uses unzip on Linux (7-zip on Windows by default) and wget to do the downloading. It uses Section 4.7.2 to load in the data.

It uses the following control tables `tiger.loader_platform`, `tiger.loader_variables`, and `tiger.loader_lookuptables` to control the process and different OS shell syntax variations.

1. `loader_variables` keeps track of various variables such as census site, year, data and staging schemas
2. `loader_platform` profiles of various platforms and where the various executables are located. Comes with windows and linux/unix. More can be added.
3. `loader_lookuptables` each record defines a kind of table (state, county), whether to process records in it and how to load them in. Defines the steps to import data, stage data, add, removes columns, indexes, and constraints for each. Each table is prefixed with the state and inherits from a table in the tiger schema. e.g. creates `tiger_data.ma_faces` which inherits from `tiger.faces`

Enhanced: 2.4.1 zip code 5 tabulation area (zcta5) load step was fixed and when enabled, zcta5 data is loaded as a single table called `zcta5_all` as part of the nation script load.

Disponibilidad: 2.1.0



#### Note

If you want zip code 5 tabulation area (zcta5) to be included in your nation script load, do the following:

```
UPDATE tiger.loader_lookuptables SET load = true WHERE table_name = 'zcta510';
```



#### Note

If you were running `tiger_2010` version and you want to reload as state with newer tiger data, you'll need to for the very first load generate and run drop statements [Drop\\_Nation\\_Tables\\_Generate\\_Script](#) before you run this script.

## Ejemplos

Generate script script to load nation data Windows.

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

Generate script to load up data for Linux/Unix systems.

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

## Ver también

[Loader\\_Generate\\_Script](#), [Missing\\_Indexes\\_Generate\\_Script](#)

### 11.2.12 Missing\_Indexes\_Generate\_Script

`Missing_Indexes_Generate_Script` — Finds all tables with key columns used in geocoder joins that are missing indexes on those columns and will output the SQL DDL to define the index for those tables.

## Synopsis

```
text Missing_Indexes_Generate_Script();
```

## Descripción

Finds all tables in `tiger` and `tiger_data` schemas with key columns used in geocoder joins that are missing indexes on those columns and will output the SQL DDL to define the index for those tables. This is a helper function that adds new indexes needed to make queries faster that may have been missing during the load process. As the geocoder is improved, this function will be updated to accommodate new indexes being used. If this function outputs nothing, it means all your tables have what we think are the key indexes already in place.

Disponibilidad: 2.0.0

## Ejemplos

```
SELECT missing_indexes_generate_script();
-- output: This was run on a database that was created before many corrections were made to ←
the loading script ---
CREATE INDEX idx_tiger_county_countyfp ON tiger.county USING btree(countyfp);
CREATE INDEX idx_tiger_cousub_countyfp ON tiger.cousub USING btree(countyfp);
CREATE INDEX idx_tiger_edges_tfidr ON tiger.edges USING btree(tfidr);
CREATE INDEX idx_tiger_edges_tfidl ON tiger.edges USING btree(tfidl);
CREATE INDEX idx_tiger_zip_lookup_all_zip ON tiger.zip_lookup_all USING btree(zip);
CREATE INDEX idx_tiger_data_ma_county_countyfp ON tiger_data.ma_county USING btree(countyfp ←
);
CREATE INDEX idx_tiger_data_ma_cousub_countyfp ON tiger_data.ma_cousub USING btree(countyfp ←
);
CREATE INDEX idx_tiger_data_ma_edges_countyfp ON tiger_data.ma_edges USING btree(countyfp);
CREATE INDEX idx_tiger_data_ma_faces_countyfp ON tiger_data.ma_faces USING btree(countyfp);
```

## Ver también

[Loader\\_Generate\\_Script](#), [Install\\_Missing\\_Indexes](#)

### 11.2.13 Normalize\_Address

`Normalize_Address` — Given a textual street address, returns a composite `norm_addy` type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This function will work with just the lookup data packaged with the `tiger_geocoder` (no need for `tiger` census data).

## Synopsis

```
norm_addy normalize_address(varchar in_address);
```

## Descripción

Given a textual street address, returns a composite `norm_addy` type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This is the first step in the geocoding process to get all addresses into normalized postal form. No other data is required aside from what is packaged with the geocoder.

This function just uses the various direction/state/suffix lookup tables preloaded with the `tiger_geocoder` and located in the `tiger` schema, so it doesn't need you to download `tiger` census data or any other additional data to make use of it. You may find the need to add more abbreviations or alternative namings to the various lookup tables in the `tiger` schema.

It uses various control lookup tables located in `tiger` schema to normalize the input address.

Fields in the `norm_addy` type object returned by this function in this order where () indicates a field required by the geocoder, [] indicates an optional field:



(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip] [parsed] [zip4] [address\_alphanumeric]

Enhanced: 2.4.0 norm\_addy object includes additional fields zip4 and address\_alphanumeric.

1. address is an integer: The street number
2. predirAbbrev is varchar: Directional prefix of road such as N, S, E, W etc. These are controlled using the direction\_lookup table.
3. streetName varchar
4. streetTypeAbbrev varchar abbreviated version of street type: e.g. St, Ave, Cir. These are controlled using the street\_type\_lookup table.
5. postdirAbbrev varchar abbreviated directional suffix of road N, S, E, W etc. These are controlled using the direction\_lookup table.
6. internal varchar internal address such as an apartment or suite number.
7. location varchar usually a city or governing province.
8. stateAbbrev varchar two character US State. e.g MA, NY, MI. These are controlled by the state\_lookup table.
9. zip varchar 5-digit zipcode. e.g. 02109.
10. parsed boolean - denotes if address was formed from normalize process. The normalize\_address function sets this to true before returning the address.
11. zip4 last 4 digits of a 9 digit zip code. Availability: PostGIS 2.4.0.
12. address\_alphanumeric Full street number even if it has alpha characters like 17R. Parsing of this is better using [Pgcode\\_Normalize\\_Address](#) function. Availability: PostGIS 2.4.0.

Ejemplos

Output select fields. Use [Pprint\\_Addy](#) if you want a pretty textual output.

```
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
FROM (SELECT address, normalize_address(address) As na
 FROM addresses_to_geocode) As g;
```

| orig                                                | streetname    | streettypeabbrev |
|-----------------------------------------------------|---------------|------------------|
| 28 Capen Street, Medford, MA                        | Capen         | St               |
| 124 Mount Auburn St, Cambridge, Massachusetts 02138 | Mount Auburn  | St               |
| 950 Main Street, Worcester, MA 01610                | Main          | St               |
| 529 Main Street, Boston MA, 02129                   | Main          | St               |
| 77 Massachusetts Avenue, Cambridge, MA 02139        | Massachusetts | Ave              |
| 25 Wizard of Oz, Walford, KS 99912323               | Wizard of Oz  |                  |

Ver también

[Geocode](#), [Pprint\\_Addy](#)

11.2.14 Pgcode\_Normalize\_Address

[Pgcode\\_Normalize\\_Address](#) — Given a textual street address, returns a composite norm\_addy type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This function will work with just the lookup data packaged with the tiger\_geocoder (no need for tiger census data). Requires address\_standardizer extension.

## Synopsis

```
norm_addy pagc_normalize_address(varchar in_address);
```

## Descripción

Given a textual street address, returns a composite `norm_addy` type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This is the first step in the geocoding process to get all addresses into normalized postal form. No other data is required aside from what is packaged with the geocoder.

This function just uses the various `pagc_*` lookup tables preloaded with the `tiger_geocoder` and located in the `tiger` schema, so it doesn't need you to download tiger census data or any other additional data to make use of it. You may find the need to add more abbreviations or alternative namings to the various lookup tables in the `tiger` schema.

It uses various control lookup tables located in `tiger` schema to normalize the input address.

Fields in the `norm_addy` type object returned by this function in this order where () indicates a field required by the geocoder, [] indicates an optional field:

There are slight variations in casing and formatting over the [Normalize\\_Address](#).

Disponibilidad: 2.1.0



This method needs `address_standardizer` extension.

(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip]

The native standardaddr of `address_standardizer` extension is at this time a bit richer than `norm_addy` since its designed to support international addresses (including country). `standardaddr` equivalent fields are:

house\_num, predir, name, suftype, sufdir, unit, city, state, postcode

Enhanced: 2.4.0 `norm_addy` object includes additional fields `zip4` and `address_alphanumeric`.

1. `address` is an integer: The street number
2. `predirAbbrev` is varchar: Directional prefix of road such as N, S, E, W etc. These are controlled using the `direction_lookup` table.
3. `streetName` varchar
4. `streetTypeAbbrev` varchar abbreviated version of street type: e.g. St, Ave, Cir. These are controlled using the `street_type_lookup` table.
5. `postdirAbbrev` varchar abbreviated directional suffice of road N, S, E, W etc. These are controlled using the `direction_lookup` table.
6. `internal` varchar internal address such as an apartment or suite number.
7. `location` varchar usually a city or governing province.
8. `stateAbbrev` varchar two character US State. e.g MA, NY, MI. These are controlled by the `state_lookup` table.
9. `zip` varchar 5-digit zipcode. e.g. 02109.
10. `parsed` boolean - denotes if address was formed from normalize process. The `normalize_address` function sets this to true before returning the address.
11. `zip4` last 4 digits of a 9 digit zip code. Availability: PostGIS 2.4.0.
12. `address_alphanumeric` Full street number even if it has alpha characters like 17R. Parsing of this is better using [Pagc\\_Normalize\\_Address](#) function. Availability: PostGIS 2.4.0.

Ejemplos

Single call example

```
SELECT addy.*
FROM pagc_normalize_address('9000 E ROO ST STE 999, Springfield, CO') AS addy;
```

|                                                                               |  |              |    |            |  |                  |   |               |  |           |  |   |
|-------------------------------------------------------------------------------|--|--------------|----|------------|--|------------------|---|---------------|--|-----------|--|---|
| address                                                                       |  | predirabbrev |    | streetname |  | streettypeabbrev |   | postdirabbrev |  | internal  |  | ↔ |
| location                                                                      |  | stateabbrev  |    | zip        |  | parsed           |   |               |  |           |  |   |
| -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----- |  |              |    |            |  |                  |   |               |  |           |  |   |
| 9000                                                                          |  | E            |    | ROO        |  | ST               |   |               |  | SUITE 999 |  | ↔ |
| SPRINGFIELD                                                                   |  |              | CO |            |  |                  | t |               |  |           |  |   |

Batch call. There are currently speed issues with the way `postgis_tiger_geocoder` wraps the `address_standardizer`. These will hopefully be resolved in later editions. To work around them, if you need speed for batch geocoding to call generate a normaddy in batch mode, you are encouraged to directly call the `address_standardizer` `standardize_address` function as shown below which is similar exercise to what we did in [Normalize\\_Address](#) that uses data created in [Geocode](#).

```
WITH g AS (SELECT address, ROW((sa).house_num, (sa).predir, (sa).name
, (sa).suftype, (sa).sufdir, (sa).unit , (sa).city, (sa).state, (sa).postcode, true)::↔
norm_addy As na
FROM (SELECT address, standardize_address('tiger.pagc_lex'
, 'tiger.pagc_gaz'
, 'tiger.pagc_rules', address) As sa
FROM addresses_to_geocode) As g)
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
FROM g;
```

|                                                     |  |               |  |                  |
|-----------------------------------------------------|--|---------------|--|------------------|
| orig                                                |  | streetname    |  | streettypeabbrev |
| -----+-----+-----                                   |  |               |  |                  |
| 529 Main Street, Boston MA, 02129                   |  | MAIN          |  | ST               |
| 77 Massachusetts Avenue, Cambridge, MA 02139        |  | MASSACHUSETTS |  | AVE              |
| 25 Wizard of Oz, Walaford, KS 99912323              |  | WIZARD OF     |  |                  |
| 26 Capen Street, Medford, MA                        |  | CAPEN         |  | ST               |
| 124 Mount Auburn St, Cambridge, Massachusetts 02138 |  | MOUNT AUBURN  |  | ST               |
| 950 Main Street, Worcester, MA 01610                |  | MAIN          |  | ST               |

Ver también

[Normalize\\_Address](#), [Geocode](#)

11.2.15 Pprint\_Addy

`Pprint_Addy` — Given a `norm_addy` composite type object, returns a pretty print representation of it. Usually used in conjunction with `normalize_address`.

Synopsis

`varchar pprint_addy(norm_addy in_addy);`

Descripción

Given a `norm_addy` composite type object, returns a pretty print representation of it. No other data is required aside from what is packaged with the geocoder.

Usually used in conjunction with [Normalize\\_Address](#).

## Ejemplos

### Pretty print a single address

```
SELECT pprint_addy(normalize_address('202 East Fremont Street, Las Vegas, Nevada 89101')) ←
 As pretty_address;
 pretty_address

202 E Fremont St, Las Vegas, NV 89101
```

### Pretty print address a table of addresses

```
SELECT address As orig, pprint_addy(normalize_address(address)) As pretty_address
 FROM addresses_to_geocode;
```

| orig                                                | pretty_address                        |
|-----------------------------------------------------|---------------------------------------|
| 529 Main Street, Boston MA, 02129                   | 529 Main St, Boston MA, 02129         |
| 77 Massachusetts Avenue, Cambridge, MA 02139        | 77 Massachusetts Ave, Cambridge, MA ← |
| 28 Capen Street, Medford, MA                        | 28 Capen St, Medford, MA              |
| 124 Mount Auburn St, Cambridge, Massachusetts 02138 | 124 Mount Auburn St, Cambridge, MA ←  |
| 950 Main Street, Worcester, MA 01610                | 950 Main St, Worcester, MA 01610      |

## Ver también

[Normalize\\_Address](#)

## 11.2.16 Reverse\_Geocode

**Reverse\_Geocode** — Takes a geometry point in a known spatial ref sys and returns a record containing an array of theoretically possible addresses and an array of cross streets. If `include_strnum_range = true`, includes the street range in the cross streets.

### Synopsis

record **Reverse\_Geocode**(geometry pt, boolean include\_strnum\_range=false, geometry[] OUT intpt, norm\_addy[] OUT addy, varchar[] OUT street);

### Descripción

Takes a geometry point in a known spatial ref and returns a record containing an array of theoretically possible addresses and an array of cross streets. If `include_strnum_range = true`, includes the street range in the cross streets. `include_strnum_range` defaults to false if not passed in. Addresses are sorted according to which road a point is closest to so first address is most likely the right one.

Why do we say theoretical instead of actual addresses. The Tiger data doesn't have real addresses, but just street ranges. As such the theoretical address is an interpolated address based on the street ranges. Like for example interpolating one of my addresses returns a 26 Court St. and 26 Court Sq., though there is no such place as 26 Court Sq. This is because a point may be at a corner of 2 streets and thus the logic interpolates along both streets. The logic also assumes addresses are equally spaced along a street, which of course is wrong since you can have a municipal building taking up a good chunk of the street range and the rest of the buildings are clustered at the end.

Note: Hmm this function relies on Tiger data. If you have not loaded data covering the region of this point, then hmm you will get a record filled with NULLS.

Returned elements of the record are as follows:

1. `intpt` is an array of points: These are the center line points on the street closest to the input point. There are as many points as there are addresses.
2. `addy` is an array of `norm_addy` (normalized addresses): These are an array of possible addresses that fit the input point. The first one in the array is most likely. Generally there should be only one, except in the case when a point is at the corner of 2 or 3 streets, or the point is somewhere on the road and not off to the side.
3. `street` an array of `varchar`: These are cross streets (or the street) (streets that intersect or are the street the point is projected to be on).

Enhanced: 2.4.1 if optional `zcta5` dataset is loaded, the `reverse_geocode` function can resolve to state and zip even if the specific state data is not loaded. Refer to [Loader\\_Generate\\_Nation\\_Script](#) for details on loading `zcta5` data.

Disponibilidad: 2.0.0

## Ejemplos

Example of a point at the corner of two streets, but closest to one. This is approximate location of MIT: 77 Massachusetts Ave, Cambridge, MA 02139 Note that although we don't have 3 streets, PostgreSQL will just return null for entries above our upper bound so safe to use. This includes street ranges

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2, pprint_addy(r.addy[3]) ←
 As st3,
 array_to_string(r.street, ',') As cross_streets
FROM reverse_geocode(ST_GeomFromText('POINT(-71.093902 42.359446)',4269),true) As r ←
;
```

```

result

 st1 | st2 | st3 | cross_streets
-----+-----+-----+-----
67 Massachusetts Ave, Cambridge, MA 02139 | | | 67 - 127 Massachusetts Ave, 32 - 88 ↩
 Vassar St

```

Here we choose not to include the address ranges for the cross streets and picked a location really really close to a corner of 2 streets thus could be known by two different addresses.

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2,
pprint_addy(r.addy[3]) As st3, array_to_string(r.street, ',') As cross_str
FROM reverse_geocode(ST_GeomFromText('POINT(-71.06941 42.34225)',4269)) As r;
```

```

result

 st1 | st2 | st3 | cross_str
-----+-----+-----+-----
5 Bradford St, Boston, MA 02118 | 49 Waltham St, Boston, MA 02118 | | Waltham St

```

For this one we reuse our geocoded example from **Geocode** and we only want the primary address and at most 2 cross streets.

```
SELECT actual_addr, lon, lat, pprint_addy((rg).addy[1]) As int_addr1,
 (rg).street[1] As cross1, (rg).street[2] As cross2
FROM (SELECT address As actual_addr, lon, lat,
 reverse_geocode(ST_SetSRID(ST_Point(lon,lat),4326)) As rg
 FROM addresses_to_geocode WHERE rating
> -1) As foo;
```

```

actual_addr | lon | lat | ↵
int_addr1 | | | ↵
cross2 | | | ↵

```

|                                                     |                                        |
|-----------------------------------------------------|----------------------------------------|
| 529 Main Street, Boston MA, 02129                   | -71.07181   42.38359   527 Main St, ↩  |
| Boston, MA 02129   Medford St                       |                                        |
| 77 Massachusetts Avenue, Cambridge, MA 02139        | -71.09428   42.35988   77 ↩            |
| Massachusetts Ave, Cambridge, MA 02139   Vassar St  |                                        |
| 26 Capen Street, Medford, MA                        | -71.12377   42.41101   9 Edison Ave, ↩ |
| Medford, MA 02155   Capen St                        | Tesla Ave                              |
| 124 Mount Auburn St, Cambridge, Massachusetts 02138 | -71.12304   42.37328   3 University ↩  |
| Rd, Cambridge, MA 02138   Mount Auburn St           |                                        |
| 950 Main Street, Worcester, MA 01610                | -71.82368   42.24956   3 Maywood St, ↩ |
| Worcester, MA 01603   Main St                       | Maywood Pl                             |

Ver también

[Pprint\\_Addy](#), [Missing\\_Indexes\\_Generate\\_Script](#)

11.2.17 Topology\_Load\_Tiger

Topology\_Load\_Tiger — Loads a defined region of tiger data into a PostGIS Topology and transforming the tiger data to spatial reference of the topology and snapping to the precision tolerance of the topology.

Synopsis

text **Topology\_Load\_Tiger**(varchar topo\_name, varchar region\_type, varchar region\_id);

Descripción

Loads a defined region of tiger data into a PostGIS Topology. The faces, nodes and edges are transformed to the spatial reference system of the target topology and points are snapped to the tolerance of the target topology. The created faces, nodes, edges maintain the same ids as the original Tiger data faces, nodes, edges so that datasets can be in the future be more easily reconciled with tiger data. Returns summary details about the process.

This would be useful for example for redistricting data where you require the newly formed polygons to follow the center lines of streets and for the resulting polygons not to overlap.



**Note** This function relies on Tiger data as well as the installation of the PostGIS topology module. For more information, refer to Chapter 8 and Section 2.2.3. If you have not loaded data covering the region of interest, then no topology records will be created. This function will also fail if you have not created a topology using the topology functions.



**Note** Most topology validation errors are a result of tolerance issues where after transformation the edges points don't quite line up or overlap. To remedy the situation you may want to increase or lower the precision if you get topology validation failures.

Required arguments:

1. topo\_name The name of an existing PostGIS topology to load data into.
2. region\_type The type of bounding region. Currently only place and county are supported. Plan is to have several more. This is the table to look into to define the region bounds. e.g tiger.place, tiger.county

3. `region_id` This is what TIGER calls the geoid. It is the unique identifier of the region in the table. For place it is the `plcidfp` column in `tiger.place`. For county it is the `cntyidfp` column in `tiger.county`

Disponibilidad: 2.0.0

### Example: Boston, Massachusetts Topology

Create a topology for Boston, Massachusetts in Mass State Plane Feet (2249) with tolerance 0.25 feet and then load in Boston city tiger faces, edges, nodes.

```
SELECT topology.CreateTopology('topo_boston', 2249, 0.25);
createtopology

15
-- 60,902 ms ~ 1 minute on windows 7 desktop running 9.1 (with 5 states tiger data loaded)
SELECT tiger.topology_load_tiger('topo_boston', 'place', '2507000');
-- topology_loader_tiger --
29722 edges holding in temporary. 11108 faces added. 1875 edges of faces added. 20576 ↔
nodes added.
19962 nodes contained in a face. 0 edge start end corrected. 31597 edges added.

-- 41 ms --
SELECT topology.TopologySummary('topo_boston');
-- topologysummary--
Topology topo_boston (15), SRID 2249, precision 0.25
20576 nodes, 31597 edges, 11109 faces, 0 topogeoms in 0 layers

-- 28,797 ms to validate yeh returned no errors --
SELECT * FROM
 topology.ValidateTopology('topo_boston');

 error | id1 | id2
-----+-----+-----
```

### Example: Suffolk, Massachusetts Topology

Create a topology for Suffolk, Massachusetts in Mass State Plane Meters (26986) with tolerance 0.25 meters and then load in Suffolk county tiger faces, edges, nodes.

```
SELECT topology.CreateTopology('topo_suffolk', 26986, 0.25);
-- this took 56,275 ms ~ 1 minute on Windows 7 32-bit with 5 states of tiger loaded
-- must have been warmed up after loading boston
SELECT tiger.topology_load_tiger('topo_suffolk', 'county', '25025');
-- topology_loader_tiger --
36003 edges holding in temporary. 13518 faces added. 2172 edges of faces added.
24761 nodes added. 24075 nodes contained in a face. 0 edge start end corrected. 38175 ↔
edges added.

-- 31 ms --
SELECT topology.TopologySummary('topo_suffolk');
-- topologysummary--
Topology topo_suffolk (14), SRID 26986, precision 0.25
24761 nodes, 38175 edges, 13519 faces, 0 topogeoms in 0 layers

-- 33,606 ms to validate --
SELECT * FROM
 topology.ValidateTopology('topo_suffolk');

 error | id1 | id2
-----+-----+-----
```

```
coincident nodes | 81045651 | 81064553
edge crosses node | 81045651 | 85737793
edge crosses node | 81045651 | 85742215
edge crosses node | 81045651 | 620628939
edge crosses node | 81064553 | 85697815
edge crosses node | 81064553 | 85728168
edge crosses node | 81064553 | 85733413
```

#### Ver también

[CreateTopology](#), [CreateTopoGeom](#), [TopologySummary](#), [ValidateTopology](#)

### 11.2.18 Set\_Geocode\_Setting

`Set_Geocode_Setting` — Sets a setting that affects behavior of geocoder functions.

#### Synopsis

```
text Set_Geocode_Setting(text setting_name, text setting_value);
```

#### Descripción

Sets value of specific setting stored in `tiger.geocode_settings` table. Settings allow you to toggle debugging of functions. Later plans will be to control rating with settings. Current list of settings are listed in [Get\\_Geocode\\_Setting](#).

Disponibilidad: 2.1.0

#### Example return debugging setting

If you run [Geocode](#) when this function is true, the NOTICE log will output timing and queries.

```
SELECT set_geocode_setting('debug_geocode_address', 'true') As result;
result

true
```

#### Ver también

[Get\\_Geocode\\_Setting](#)



## Chapter 12

# PostGIS Special Functions Index

### 12.1 PostGIS Aggregate Functions

The functions below are spatial aggregate functions that are used in the same way as SQL aggregate function such as `sum` and `average`.

- **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
  - **ST\_3DUnion** - Perform 3D union.
  - **ST\_AsFlatGeobuf** - Return a FlatGeobuf representation of a set of rows.
  - **ST\_AsGeobuf** - Return a Geobuf representation of a set of rows.
  - **ST\_AsMVT** - Aggregate function returning a MVT representation of a set of rows.
  - **ST\_ClusterIntersecting** - Aggregate function that clusters input geometries into connected sets.
  - **ST\_ClusterWithin** - Aggregate function that clusters input geometries by separation distance.
  - **ST\_CoverageUnion** - Computes the union of a set of polygons forming a coverage by removing shared edges.
  - **ST\_Extent** - Aggregate function that returns the bounding box of geometries.
  - **ST\_GeomCollFromText** - Creates a GeometryCollection or Multi\* geometry from a set of geometries.
  - **ST\_MakeLine** - Crea una cadena de línea desde geometrías de punto, multipunto o de línea.
  - **ST\_MemUnion** - Aggregate function which unions geometries in a memory-efficient but slower way
  - **ST\_Polygonize** - Computes a collection of polygons formed from the linework of a set of geometries.
  - **ST\_SameAlignment** - Returns true if rasters have same skew, scale, spatial ref, and offset (pixels can be put on same grid without cutting into pixels) and false if they don't with notice detailing issue.
  - **ST\_Union** - Computes a geometry representing the point-set union of the input geometries.
  - **TopoElementArray\_Agg** - Devuelve un `topoelementarray` para un conjunto de `element_id`, matriz de tipo (`topoelements`).
-

## 12.2 PostGIS Window Functions

The functions below are spatial window functions that are used in the same way as SQL window functions such as `row_number()`, `lead()`, and `lag()`. They must be followed by an `OVER()` clause.

- **ST\_ClusterDBSCAN** - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- **ST\_ClusterIntersectingWin** - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
- **ST\_ClusterKMeans** - Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST\_ClusterWithinWin** - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- **ST\_CoverageInvalidEdges** - Window function that finds locations where polygons fail to form a valid coverage.
- **ST\_CoverageSimplify** - Window function that simplifies the edges of a polygonal coverage.

## 12.3 PostGIS SQL-MM Compliant Functions

The functions given below are PostGIS functions that conform to the SQL/MM 3 standard

- **ST\_3DArea** - Computes area of 3D surface geometries. Will return 0 for solids. Descripción Disponibilidad: 2.1.0 This method needs SFCGAL backend. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 8.1, 10.5 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- **ST\_3DDWithin** - Tests if two 3D geometries are within a given 3D distance Description Returns true if the 3D distance between two geometry values is no larger than distance distance\_of\_srid. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense the source geometries must be in the same coordinate system (have the same SRID). This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This method implements the SQL/MM specification. SQL-MM ? Availability: 2.0.0
- **ST\_3DDifference** - Perform 3D difference Descripción Returns that part of geom1 that is not part of geom2. Disponibilidad: 2.2.0 This method needs SFCGAL backend. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- **ST\_3DDistance** - Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units. Descripción Returns the 3-dimensional minimum cartesian distance between two geometries in projected units (spatial ref units). This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3 Disponibilidad: 2.0.0 Changed: 2.2.0 - In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z. Changed: 3.0.0 - SFCGAL version removed
- **ST\_3DIntersection** - Perform 3D intersection Descripción Return a geometry that is the shared portion between geom1 and geom2. Disponibilidad: 2.1.0 This method needs SFCGAL backend. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- **ST\_3DIntersects** - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area) Description Overlaps, Touches, Within all imply spatial intersection. If any of the aforementioned returns true, then the geometries also spatially intersect. Disjoint implies false for spatial intersection. This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs. Availability: 2.0.0 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN). This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

- **ST\_3DLength** - Returns the 3D length of a linear geometry. Descripción Returns the 3-dimensional or 2-dimensional length of the geometry if it is a LineString or MultiLineString. For 2-d lines it will just return the 2-d length (same as ST\_Length and ST\_Length2D) This function supports 3d and will not drop the z-index. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 7.1, 10.3 Changed: 2.0.0 In prior versions this used to be called ST\_Length3D
- **ST\_3DPerimeter** - Returns the 3D perimeter of a polygonal geometry. Descripción Returns the 3-dimensional perimeter of the geometry, if it is a polygon or multi-polygon. If the geometry is 2-dimensional, then the 2-dimensional perimeter is returned. This function supports 3d and will not drop the z-index. This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.1, 10.5 Changed: 2.0.0 In prior versions this used to be called ST\_Perimeter3D
- **ST\_3DUnion** - Perform 3D union. Descripción Disponibilidad: 2.2.0 Availability: 3.3.0 aggregate variant was added This method needs SFCGAL backend. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN). Aggregate variant: returns a geometry that is the 3D union of a rowset of geometries. The ST\_3DUnion() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.
- **ST\_AddEdgeModFace** - Añada un nuevo borde y, si al hacerlo, divide una cara, modifica la cara original y añade una nueva cara. Descripción Añade un nuevo borde y, si al hacerlo, se divide una cara, modifica la cara original y añade una nueva. Si es posible, la nueva cara se creará en el lado izquierdo del nuevo borde. Esto no será posible si la cara del lado izquierdo necesita ser Universe face (sin límites). Devuelve el identificador del borde recientemente añadido. Actualiza todos los bordes unidos y relaciones en consecuencia existentes. Si cualquier argumento es nulo, los nodos dados son desconocidos (ya deben existir en la tabla node del esquema de topología), el acurve no es un LINESTRING, el anode y anothernode no son el punto de inicio y final de acurve entonces un error es lanzado. Si el sistema de referencia espacial (SRID) de la geometría acurve no es el mismo que la topología se lanza una excepción. Disponibilidad: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalle de Rutina: X.3.13
- **ST\_AddEdgeNewFaces** - Agrega un nuevo borde y, si al hacerlo divide una cara, se elimina la cara original y es reemplazada con dos nuevas caras. Descripción Agrega un nuevo borde y, si al hacerlo divide una cara, se elimina la cara original y es reemplazada con dos nuevas caras. Devuelve el identificador del borde recientemente agregado. Actualiza todos los bordes unidos y relaciones en consecuencia existentes. Si cualquier argumento es nulo, los nodos dados son desconocidos (ya deben existir en la tabla node del esquema de topología), el acurve no es un LINESTRING, el anode y anothernode no son el punto de inicio y final de acurve entonces un error es lanzado. Si el sistema de referencia espacial (SRID) de la geometría acurve no es el mismo que la topología se lanza una excepción. Disponibilidad: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.12
- **ST\_AddIsoEdge** - Agrega un borde aislado definido por la geometría alinestring a una topología que conecta dos nodos aislados existentes anode y anothernode y devuelve el identificador de borde del nuevo borde. Descripción Agrega un borde aislado definido por la geometría alinestring a una topología que conecta dos nodos aislados existentes anode y anothernode y devuelve el identificador de borde del nuevo borde. Si el sistema de referencia espacial (SRID) de la geometría alinestring no es el mismo que la topología, cualquiera de los argumentos de entrada son nulos, o los nodos se contienen en más de una cara, o los nodos son el inicio o fin de los nodos de un borde existente, entonces una excepción es lanzada. Si el alinestring no está dentro de la cara de la cara a la que pertenece anode y anothernode, entonces una excepción es lanzada. Si el anode y anothernode no son los puntos de inicio y final de la alinestring entonces una excepción es lanzada. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de rutina: X.3.4
- **ST\_AddIsoNode** - Agrega un nodo aislado a una cara de una topología y devuelve el identificador de nodo del nuevo nodo. Si la cara es nula, el nodo es creado de todas maneras. Descripción Agrega un nodo aislado con la localización del punto apoint a una cara existente con identificador de cara aface a una topología atopolology y devuelve el identificador de nodo de el nuevo nodo. Si el sistema de referencia espacial (SRID) de la geometría de punto no es el mismo que el de la topología, el apoint no es una geometría de punto, el punto es nulo, o el punto intersecta un borde existente (incluso en los límites) entonces una excepción es lanzada. Si el punto ya existe como un nodo, se produce una excepción. Si aface no es nulo y el apoint no está dentro de la cara, entonces una excepción es lanzada. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Net Rutinas: X+1.3.1
- **ST\_Area** - Returns the area of a polygonal geometry. Descripción Returns the area of a polygonal geometry. For geometry types a 2D Cartesian (planar) area is computed, with units specified by the SRID. For geography types by default area is determined on a spheroid with units in square meters. To compute the area using the faster but less accurate spherical model use

`ST_Area(geog,false)`. Enhanced: 2.0.0 - support for 2D polyhedral surfaces was introduced. Enhanced: 2.2.0 - measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires PROJ >= 4.9.0 to take advantage of the new feature. Changed: 3.0.0 - does not depend on SFCGAL anymore. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 8.1.2, 9.5.3 This function supports Polyhedral surfaces. For polyhedral surfaces, only supports 2D polyhedral surfaces (not 2.5D). For 2.5D, may give a non-zero answer, but only for the faces that sit completely in XY plane.

- ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data. Descripción Returns the OGC/ISO Well-Known Binary (WKB) representation of the geometry. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding, either little-endian ('NDR') or big-endian ('XDR'). WKB format is useful to read geometry data from the database and maintaining full numeric precision. This avoids the precision rounding that can happen with text formats such as WKT. To perform the inverse conversion of WKB to PostGIS geometry use . The OGC/ISO WKB format does not include the SRID. To get the EWKB format which does include the SRID use The default behavior in PostgreSQL 9.0 has been changed to output bytea in hex encoding. If your GUI tools require the old behavior, then SET bytea\_output='escape' in your database. Mejorado: 2.0.0 soporte para superficies poliédricas, triángulos y TIN fue introducida. Enhanced: 2.0.0 support for higher coordinate dimensions was introduced. Enhanced: 2.0.0 support for specifying endian with geography was introduced. Availability: 1.5.0 geography support was introduced. Changed: 2.0.0 Inputs to this function can not be unknown -- must be geometry. Constructs such as `ST_AsBinary('POINT(1 2)')` are no longer valid and you will get an `st_asbinary(unknown)` is not unique error. Code like that needs to be changed to `ST_AsBinary('POINT(1 2)::geometry')`. If that is not possible, then install legacy.sql. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.37 This method supports Circular Strings and Curves. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN). This function supports 3d and will not drop the z-index.
- ST\_AsGML** - Return the geometry as a GML version 2 or 3 element. Descripción Return the geometry as a Geography Markup Language (GML) element. The version parameter, if specified, may be either 2 or 3. If no version parameter is specified then the default is assumed to be 2. The maxdecimaldigits argument may be used to reduce the maximum number of decimal places used in output (defaults to 15). Using the maxdecimaldigits parameter can cause output geometry to become invalid. To avoid this use with a suitable gridsize first. GML 2 refer to 2.1.2 version, GML 3 to 3.1.1 version The 'options' argument is a bitfield. It could be used to define CRS output type in GML output, and to declare data as lat/lon: 0: GML Short CRS (e.g EPSG:4326), default value 1: GML Long CRS (e.g urn:ogc:def:crs:EPSG::4326) 2: For GML 3 only, remove srsDimension attribute from output. 4: For GML 3 only, use <LineString> rather than <Curve> tag for lines. 16: Declare that datas are lat/lon (e.g srid=4326). Default is to assume that data are planars. This option is useful for GML 3.1.1 output only, related to axis order. So if you set it, it will swap the coordinates so order is lat lon instead of database lon lat. 32: Output the box of the geometry (envelope). The 'namespace prefix' argument may be used to specify a custom namespace prefix or no prefix (if empty). If null or omitted 'gml' prefix is used Disponibilidad: 1.3.2 Availability: 1.5.0 geography support was introduced. Enhanced: 2.0.0 prefix support was introduced. Option 4 for GML3 was introduced to allow using LineString instead of Curve tag for lines. GML3 Support for Polyhedral surfaces and TINS was introduced. Option 32 was introduced to output the box. Changed: 2.0.0 use default named args Enhanced: 2.1.0 id support was introduced, for GML 3. Only version 3+ of ST\_AsGML supports Polyhedral Surfaces and TINS. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 17.2 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ST\_AsText** - Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata. Descripción Returns the OGC Well-Known Text (WKT) representation of the geometry/geography. The optional maxdecimaldigits argument may be used to limit the number of digits after the decimal point in output ordinates (defaults to 15). To perform the inverse conversion of WKT representation to PostGIS geometry use . The standard OGC WKT representation does not include the SRID. To include the SRID as part of the output representation, use the non-standard PostGIS function The textual representation of numbers in WKT may not maintain full floating-point precision. To ensure full accuracy for data storage or transport it is best to use Well-Known Binary (WKB) format (see and maxdecimaldigits). Using the maxdecimaldigits parameter can cause output geometry to become invalid. To avoid this use with a suitable gridsize first. Availability: 1.5 - support for geography was introduced. Enhanced: 2.5 - optional parameter precision introduced. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.25 This method supports Circular Strings and Curves.
- ST\_Boundary** - Devuelve el cierre del limite combinatorio de esta geometría. Descripción Devuelve el cierre del limite combinatorio de esta geometría. El limite combinatorio esta definido como se describe en la sección 3.12.3.2 de la especificación

OGC. Ya que el resultado de esta función es un cerco, y por lo tanto topológicamente cerrado, el límite resultante puede ser representado utilizando geometrías primitivas como se discute en la especificación OGC en la sección 3.12.2. Realizado por el módulo de GEOS Anterior a la versión 2.0.0, esta función lanza una excepción si se utiliza con GEOMETRYCOLLECTION. Desde la versión 2.0.0 y superiores devolverá NULL en lugar de la excepción (entrada no soportada). This method implements the OGC Simple Features Implementation Specification for SQL 1.1. OGC SPEC s2.1.1.1 This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.17 This function supports 3d and will not drop the z-index. Mejorado: 2.1.0 Se ha introducido soporte para Triangle Changed: 3.2.0 support for TIN, does not use geos, does not linearize curves

- ST\_Buffer** - Computes a geometry covering all points within a given distance from a geometry. Descripción Computes a POLYGON or MULTIPOLYGON that represents all points whose distance from a geometry/geography is less than or equal to a given distance. A negative distance shrinks the geometry rather than expanding it. A negative distance may shrink a polygon completely, in which case POLYGON EMPTY is returned. For points and lines negative distances always return empty results. For geometry, the distance is specified in the units of the Spatial Reference System of the geometry. For geography, the distance is specified in meters. The optional third parameter controls the buffer accuracy and style. The accuracy of circular arcs in the buffer is specified as the number of line segments used to approximate a quarter circle (default is 8). The buffer style can be specified by providing a list of blank-separated key=value pairs as follows: 'quad\_segs=#' : number of line segments used to approximate a quarter circle (default is 8). 'endcap=round|flat|square' : endcap style (defaults to "round"). 'butt' is accepted as a synonym for 'flat'. 'join=round|mitre|bevel' : join style (defaults to "round"). 'miter' is accepted as a synonym for 'mitre'. 'mitre\_limit=#.#' : mitre ratio limit (only affects mitered join style). 'miter\_limit' is accepted as a synonym for 'mitre\_limit'. 'side=both|left|right' : 'left' or 'right' performs a single-sided buffer on the geometry, with the buffered side relative to the direction of the line. This is only applicable to LINESTRING geometry and does not affect POINT or POLYGON geometries. By default end caps are square. For geography this is a thin wrapper around the geometry implementation. It determines a planar spatial reference system that best fits the bounding box of the geography object (trying UTM, Lambert Azimuthal Equal Area (LAEA) North/South pole, and finally Mercator ). The buffer is computed in the planar space, and then transformed back to WGS84. This may not produce the desired behavior if the input object is much larger than a UTM zone or crosses the dateline Buffer output is always a valid polygonal geometry. Buffer can handle invalid inputs, so buffering by distance 0 is sometimes used as a way of repairing invalid polygons. can also be used for this purpose. Buffering is sometimes used to perform a within-distance search. For this use case it is more efficient to use . This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry. Enhanced: 2.5.0 - ST\_Buffer geometry support was enhanced to allow for side buffering specification side=both|left|right. Availability: 1.5 - ST\_Buffer was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added. Realizado por el módulo GEOS. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.30
- ST\_Centroid** - Returns the geometric center of a geometry. Descripción Computes a point which is the geometric center of mass of a geometry. For [MULTI]POINTS, the centroid is the arithmetic mean of the input coordinates. For [MULTI]LINESTRINGs, the centroid is computed using the weighted length of each line segment. For [MULTI]POLYGONs, the centroid is computed in terms of area. If an empty geometry is supplied, an empty GEOMETRYCOLLECTION is returned. If NULL is supplied, NULL is returned. If CIRCULARSTRING or COMPOUNDCURVE are supplied, they are converted to linestring with CurveToLine first, then same than for LINESTRING For mixed-dimension input, the result is equal to the centroid of the component Geometries of highest dimension (since the lower-dimension geometries contribute zero "weight" to the centroid). Note that for polygonal geometries the centroid does not necessarily lie in the interior of the polygon. For example, see the diagram below of the centroid of a C-shaped polygon. To construct a point guaranteed to lie in the interior of a polygon use . New in 2.3.0 : supports CIRCULARSTRING and COMPOUNDCURVE (using CurveToLine) Availability: 2.4.0 support for geography was introduced. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5
- ST\_ChangeEdgeGeom** - Cambia la forma de un borde sin afectar la estructura de la topología. Descripción Cambia la forma de un borde sin afectar la estructura de la topología. If any arguments are null, the given edge does not exist in the edge table of the topology schema, the acurve is not a LINESTRING, or the modification would change the underlying topology then an error is thrown. Si el sistema de referencia espacial (SRID) de la geometría acurve no es el mismo que la topología se lanza una excepción. Si el nuevo acurve no es simple, entonces un error es lanzado. Si al mover el borde de la vieja a la nueva posición golpease un obstáculo entonces se produce un error. Disponibilidad: 1.1.0 Mejorado: 2.0.0 agrega aplicación de consistencia topológica This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalle de Rutina X.3.6
- ST\_Contains** - Tests if every point of B lies in A, and their interiors have a point in common Description Returns TRUE if geometry A contains geometry B. A contains B if and only if all points of B lie inside (i.e. in the interior or boundary of)



A (or equivalently, no points of B lie in the exterior of A), and the interiors of A and B have at least one point in common. In mathematical terms:  $ST\_Contains(A, B) \Leftrightarrow (A \cap B = B) \wedge (Int(A) \cap Int(B) \neq \emptyset)$  The contains relationship is reflexive: every geometry contains itself. (In contrast, in the predicate a geometry does not properly contain itself.) The relationship is antisymmetric: if  $ST\_Contains(A, B) = true$  and  $ST\_Contains(B, A) = true$ , then the two geometries must be topologically equal ( $ST\_Equals(A, B) = true$ ).  $ST\_Contains$  is the converse of  $ST\_Within$ . So,  $ST\_Contains(A, B) = ST\_Within(B, A)$ . Because the interiors must have a common point, a subtlety of the definition is that polygons and lines do not contain lines and points lying fully in their boundary. For further details see Subtleties of OGC Covers, Contains, Within. The predicate provides a more inclusive relationship. This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function  $_ST\_Contains$ . Performed by the GEOS module Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Do not use this function with invalid geometries. You will get unexpected results. NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - same as  $within(geometry B, geometry A)$  This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

- ST\_ConvexHull** - Computes the convex hull of a geometry. Descripción Computes the convex hull of a geometry. The convex hull is the smallest convex geometry that encloses all geometries in the input. One can think of the convex hull as the geometry obtained by wrapping an rubber band around a set of geometries. This is different from a concave hull which is analogous to "shrink-wrapping" the geometries. A convex hull is often used to determine an affected area based on a set of point observations. In the general case the convex hull is a Polygon. The convex hull of two or more collinear points is a two-point LineString. The convex hull of one or more identical points is a Point. This is not an aggregate function. To compute the convex hull of a set of geometries, use to aggregate them into a geometry collection (e.g.  $ST\_ConvexHull(ST\_Collect(geom))$ ). Realizado por el módulo de GEOS This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.16 This function supports 3d and will not drop the z-index.
- ST\_CoordDim** - Devuelve la dimensión de las coordenadas del valor de ST\_Geometry. Descripción Devuelve la dimensión de las coordenadas del valor de ST\_Geometry. Es el alias de conforme a MM This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 5.1.3 This method supports Circular Strings and Curves. This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ST\_CreateTopoGeo** - Agrega una colección de geometrías a una topología vacía dada y devuelve un mensaje que detalla el éxito. Descripción Agrega una colección de geometrías a una topología vacía dada y devuelve un mensaje que detalla el éxito. Útil para rellenar una topología vacía. Disponibilidad: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de la rutina -- X.3.18
- ST\_Crosses** - Tests if two geometries have some, but not all, interior points in common Description Compares two geometry objects and returns true if their intersection "spatially crosses"; that is, the geometries have some, but not all interior points in common. The intersection of the interiors of the geometries must be non-empty and must have dimension less than the maximum dimension of the two input geometries, and the intersection of the two geometries must not equal either geometry. Otherwise, it returns false. The crosses relation is symmetric and irreflexive. In mathematical terms:  $ST\_Crosses(A, B) \Leftrightarrow (dim(Int(A) \cap Int(B)) < \max(dim(Int(A)), dim(Int(B))) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$  Geometries cross if their DE-9IM Intersection Matrix matches: T\*T\*\*\*\*\* for Point/Line, Point/Area, and Line/Area situations T\*\*\*\*\*T\*\* for Line/Point, Area/Point, and Area/Line situations 0\*\*\*\*\* for Line/Line situations the result is false for Point/Point and Area/Area situations The OpenGIS Simple Features Specification defines this predicate only for Point/Line, Point/Area, Line/Line, and Line/Area situations. JTS / GEOS extends the definition to apply to Line/Point, Area/Point and Area/Line situations as well. This makes the relation symmetric. This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.13.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.29
- ST\_CurveToLine** - Converts a geometry containing curves to a linear geometry. Descripción Converts a CIRCULAR STRING to regular LINESTRING or CURVEPOLYGON to POLYGON or MULTISURFACE to MULTIPOLYGON. Useful for outputting to devices that can't support CIRCULARSTRING geometry types Converts a given geometry to a linear geometry. Each curved geometry or segment is converted into a linear approximation using the given `tolerance` and options (32 segments per quadrant and no options by default). The `tolerance\_type` argument determines interpretation of the `tolerance` argument. It can take the following values: 0 (default): Tolerance is max segments per quadrant. 1: Tolerance is max-deviation of line from curve, in source units. 2: Tolerance is max-angle, in radians, between generating radii. The `flags` argument is

a bitfield. 0 by default. Supported bits are: 1: Symmetric (orientation independent) output. 2: Retain angle, avoids reducing angles (segment lengths) when producing symmetric output. Has no effect when Symmetric flag is off. Availability: 1.3.0 Enhanced: 2.4.0 added support for max-deviation and max-angle tolerance, and for symmetric output. Enhanced: 3.0.0 implemented a minimum number of segments per linearized arc to prevent topological collapse. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 7.1.7 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves.

- ST\_Difference** - Computes a geometry representing the part of geometry A that does not intersect geometry B. Description Returns a geometry representing the part of geometry A that does not intersect geometry B. This is equivalent to `A - ST_Intersection(A,B)`. If A is completely contained in B then an empty atomic geometry of appropriate type is returned. This is the only overlay function where input order matters. `ST_Difference(A, B)` always returns a portion of A. If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher) Performed by the GEOS module Enhanced: 3.1.0 accept a `gridSize` parameter. Requires GEOS  $\geq$  3.9.0 to use the `gridSize` parameter. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.20 This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.
- ST\_Dimension** - Devuelve la dimensión de las coordenadas del valor de ST\_Geometry. Descripción La dimensión inherente del objeto Geometry, la cual debe ser menor o igual a la dimensión de coordenadas. En la Especificación OGC s2.1.1.1 - devuelve 0 para un POINT, 1 para una LINESTRING, 2 para un POLYGON, y la dimensión mayor de los componentes de una GEOMETRYCOLLECTION. Si es desconocida (geometría vacía) se devuelve null. This method implements the SQL/MM specification. SQL-MM 3: 5.1.2 Mejora: 2.0.0 se introdujeron soporte de superficies poliédricas y TIN. No lanza una excepción si se envía una geometría vacía. Anterior a la versión 2.0.0, esta función lanzaba una excepción si se enviaba una geometría vacía. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ST\_Disjoint** - Tests if two geometries have no points in common Description Returns true if two geometries are disjoint. Geometries are disjoint if they have no point in common. If any other spatial relationship is true for a pair of geometries, they are not disjoint. Disjoint implies that is false. In mathematical terms:  $ST\_Disjoint(A, B) \Leftrightarrow A \cap B = \emptyset$  Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Performed by the GEOS module This function call does not use indexes. A negated predicate can be used as a more performant alternative that uses indexes: `ST_Disjoint(A,B) = NOT ST_Intersects(A,B)` NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 //s2.1.13.3 - a.Relate(b, 'FF\*FF\*\*\*\*') This method implements the SQL/MM specification. SQL-MM 3: 5.1.26
- ST\_Distance** - Returns the distance between two geometry or geography values. Descripción For types returns the minimum 2D Cartesian (planar) distance between two geometries, in projected units (spatial ref units). For types defaults to return the minimum geodesic distance between two geographies in meters, compute on the spheroid determined by the SRID. If `use_spheroid` is false, a faster spherical calculation is used. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 5.1.23 This method supports Circular Strings and Curves. Availability: 1.5.0 geography support was introduced in 1.5. Speed improvements for planar to better handle large or many vertex geometries Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details. Enhanced: 2.1.0 - support for curved geometries was introduced. Enhanced: 2.2.0 - measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires PROJ  $\geq$  4.9.0 to take advantage of the new feature. Changed: 3.0.0 - does not depend on SFCGAL anymore.
- ST\_EndPoint** - Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString. Descripción Devuelve el primer punto de una geometría LINESTRING o CIRCULARLINESTRING como un POINT o NULL si el parámetro de entrada no es un LINESTRING o CIRCULARLINESTRING. This method implements the SQL/MM specification. SQL-MM 3: 7.1.4 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. Cambiado: 2.0.0 ya no funciona con multilinestrings de geometrías simples. En versiones anteriores de PostGIS -- una línea simple multilinestring funciona sin problemas con esta función y devuelve el punto inicial. En la versión 2.0.0 simplemente devuelve NULL como con cualquier multilinestring. La antigua versión era una función sin documentar, pero la gente que asumía que tenía sus datos almacenados en LINESTRING pueden experimentar este comportamiento ahora de resultado NULL en la versión 2.0.
- ST\_Envelope** - Devuelve una geometría que representa la caja en doble precisión (float8) de la geometría dada. Descripción Devuelve una geometría que representa la caja mínima en doble precisión (float8) de la geometría dada. El polígono definido

por las esquinas de la caja ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)). (PostGIS añadirá las coordenadas ZMIN/ZMAX también). Algunos casos particulares (líneas verticales, puntos) devolverán una geometría de dimension menor que POLYGON, por ejemplo POINT o LINESTRING. Disponibilidad: 1.5.0 comportamiento modificado para devolver doble precisión en vez de float4. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.19

- **ST\_Equals** - Tests if two geometries include the same set of points Description Returns true if the given geometries are "topologically equal". Use this for a 'better' answer than '='. Topological equality means that the geometries have the same dimension, and their point-sets occupy the same space. This means that the order of vertices may be different in topologically equal geometries. To verify the order of points is consistent use (it must be noted ST\_OrderingEquals is a little more stringent than simply verifying order of points are the same). In mathematical terms:  $ST\_Equals(A, B) \Leftrightarrow A = B$  The following relation holds:  $ST\_Equals(A, B) \Leftrightarrow ST\_Within(A, B) \wedge ST\_Within(B, A)$  Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 This method implements the SQL/MM specification. SQL-MM 3: 5.1.24 Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal
- **ST\_ExteriorRing** - Devuelva el número de anillos interiores de una geometría poligonal. Descripción Devuelve una linestring representando el anillo exterior de una geometría tipo POLYGON. Devuelve NULL si la geometría no es un polígono. No funcionará con MULTIPOLYGON Esto no funcionara con MULTIPOLYGONs. Para MULTIPOLYGONs utilizaba junto a ST\_Dump. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1 This method implements the SQL/MM specification. SQL-MM 3: 8.2.3, 8.3.3 This function supports 3d and will not drop the z-index.
- **ST\_GMLToSQL** - Devuelve un valor específico ST\_Geometry desde una representación GML. Esto es un alias de ST\_GeomFromGML Descripción This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (excepto para soporte de curvas). Disponibilidad: 1.5, requiere libxml2 1.6+ Mejora: 2.0.0 se introdujeron soporte de superficies poliédricas y TIN. Mejorada: 2.0.0 se agregó el parámetro por defecto opcional srid.
- **ST\_GeomCollFromText** - Hace una colección Geometry de la colección WKT con el SRID dado. Si no se da SRID, el valor predeterminado es 0. Descripción Hace una colección Geometry de la representación de texto conocido (WKT) con el SRID dado. Si no se da SRID, el valor predeterminado es 0. OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad Devuelve null si el WKT no es una GEOMETRYCOLLECTION Si estas completamente seguro que todas tus geometrías WKT son colecciones, no utilices esta función. Es mas lenta que ST\_GeomFromText ya que añade pasos de validación adicionales. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification.
- **ST\_GeomFromText** - Devuelve un valor específico de ST\_Geometry desde una representación "Extended Well-Known Binary" (EWKB). Descripción Construye un objeto ST\_Geometry de PostGIS desde una representación OGC "Well-Known Text" (WKT). Hay dos variantes de la función ST\_GeomFromText. El primero no toma SRID y devuelve una geometría sin sistema de referencia espacial definido (SRID = 0). La segunda toma un SRID como segundo argumento y devuelve una geometría que incluye esta SRID como parte de sus metadatos. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - la opción SRID es de la suite de conformidad. This method implements the SQL/MM specification. SQL-MM 3: 5.1.40 This method supports Circular Strings and Curves. While not OGC-compliant, is faster than ST\_GeomFromText and ST\_PointFromText. It is also easier to use for numeric coordinate values. is another option similar in speed to and is OGC-compliant, but doesn't support anything but 2D points. Cambiado: 2.0.0 En las versiones anteriores de PostGIS ST\_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') estaba permitido. Esto no está permitido ahora en PostGIS 2.0.0 para ajustarse mejor a las normas SQL/MM. Esto debería ser escrito como ST\_GeomFromText('GEOMETRYCOLLECTION EMPTY')
- **ST\_GeomFromWKB** - Crea una instancia de geometría desde la representación de una geometría en "Well-Known Binary" (WKB) y un SRID opcional. Descripción La función ST\_GeomFromWKB, toma una representación binaria "well-known" de una geometría y un ID de un Sistema de Referencia Espacial (SRID) y crea una instancia del tipo de geometría adecuado. Esta función juega un rol de "Geometry Factory" en SQL. Es un nombre alternativo para ST\_WKBToSQL. Si no se especifica SRID, el valor predeterminado es 0 (desconocido). This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2 - El SRID opcional es para el paquete de conformidad This method implements the SQL/MM specification. SQL-MM 3: 5.1.41 This method supports Circular Strings and Curves.
- **ST\_GeometryFromText** - Devuelve un valor específico de ST\_Geometry desde una representación "Well-Known Text" (WKT). Es un alias para ST\_GeomFromText Descripción This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 5.1.40



- **ST\_GeometryN** - Devuelve el tipo de geometría del valor de ST\_Geometry. Descripción Devuelve la geometría en la cual se basa si la geometría es una GEOMETRYCOLLECTION, un (MULTI)POINT, una (MULTI)LINestring, una MULTICURVE o un (MULTI)POLYGON, una POLYHEDRALSURFACE si no devuelve NULL. El índice es 1-based en la especificación OGC desde la versión 0.8.0. Versiones anteriormente implementadas era de tipo 0-based. Si quieres extraer todas las geometrías de una geometría, ST\_Dump es más eficiente y funcionará con geometrías simples. Mejorado: 2.0.0 se introdujo soporte para superficies poliédricas, Triangulos y TIN. Cambiado: 2.0.0 Versiones anteriores devuelven NULL para geometrías simples. Esto ha sido cambiado para devolver la geometría en el caso de ST\_GeometryN(...,1) . This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 9.1.5 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- **ST\_GeometryType** - Devuelve el tipo de geometría del valor de ST\_Geometry. Descripción Devuelve el tipo de geometría como una cadena de texto. Por Ejemplo: 'ST\_LineString', 'ST\_Polygon', 'ST\_MultiPolygon' etc. Esta función difiere de GeometryType(geometría) en este caso se devuelve la cadena de texto y ST delante, como el hecho de que no indicará como se mide la geometría. Mejora: 2.0.0 se introdujo soporte de superficies poliédricas. This method implements the SQL/MM specification. SQL-MM 3: 5.1.4 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces.
- **ST\_GetFaceEdges** - Devuelve un conjunto de bordes ordenados que ligan afase. Descripción Devuelve un conjunto de bordes ordenados que ligan afase. Cada salida consta de una secuencia e identificador de borde. Los números de secuencia comienzan con el valor 1. La enumeración de los bordes de cada anillo comienza desde el borde con el identificador más pequeño. El orden de los bordes sigue la regla de la izquierda (la cara enmarcada está a la izquierda de cada borde dirigido). Disponibilidad: 2.0 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.5
- **ST\_GetFaceGeometry** - Devuelve el polígono en la topología dada con el identificador de la cara especificada. Descripción Devuelve el polígono en la topología dada con el identificador de cara especificado. Construye el polígono de los bordes que componen la cara. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.16
- **ST\_InitTopoGeo** - Creates a new topology schema and registers it in the topology.topology table. Descripción This is the SQL-MM equivalent of . It lacks options for spatial reference system and tolerance. it returns a text description of the topology creation, instead of the topology id. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo y Topo-Net 3: Detalles de la rutina: X.3.17
- **ST\_InteriorRingN** - Devuelva el número de anillos interiores de una geometría poligonal. Descripción Devuelve la cadena de texto del anillo interior N del polígono. Devuelve NULL si la geometría no es un polígono o el índice N dado esta fuera de rango. Esto no funcionara con MULTIPOLYGONs. Para MULTIPOLYGONs utilizaba junto a ST\_Dump. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5 This function supports 3d and will not drop the z-index.
- **ST\_Intersection** - Computes a geometry representing the shared portion of geometries A and B. Description Returns a geometry representing the point-set intersection of two geometries. In other words, that portion of geometry A and geometry B that is shared between the two geometries. If the geometries have no points in common (i.e. are disjoint) then an empty atomic geometry of appropriate type is returned. If the optional gridSize argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher) ST\_Intersection in conjunction with is useful for clipping geometries such as in bounding box, buffer, or region queries where you only require the portion of a geometry that is inside a country or region of interest. For geography this is a thin wrapper around the geometry implementation. It first determines the best SRID that fits the bounding box of the 2 geography objects (if geography objects are within one half zone UTM but not same UTM will pick one of those) (favoring UTM or Lambert Azimuthal Equal Area (LAEA) north/south pole, and falling back on mercator in worst case scenario) and then intersection in that best fit planar spatial ref and retransforms back to WGS84 geography. This function will drop the M coordinate values if present. If working with 3D geometries, you may want to use SFCGAL based which does a proper 3D intersection for 3D geometries. Although this function works with Z-coordinate, it does an averaging of Z-Coordinate. Performed by the GEOS module Enhanced: 3.1.0 accept a gridSize parameter Requires GEOS >= 3.9.0 to use the gridSize parameter Changed: 3.0.0 does not depend on SFCGAL. Availability: 1.5 support for geography data type was introduced. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.18 This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

- ST\_Intersects** - Tests if two geometries intersect (they have at least one point in common) Description Returns true if two geometries intersect. Geometries intersect if they have any point in common. For geography, a distance tolerance of 0.00001 meters is used (so points that are very close are considered to intersect). In mathematical terms:  $ST\_Intersects(A, B) \Leftrightarrow A \cap B \neq \emptyset$  Geometries intersect if their DE-9IM Intersection Matrix matches one of: T\*\*\*\*\* \*T\*\*\*\*\* \*\*T\*\*\*\*\* \*\*T\*\*\*\*\* Spatial intersection is implied by all the other spatial relationship tests, except , which tests that geometries do NOT intersect. This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. Changed: 3.0.0 SFCGAL version removed and native support for 2D TINs added. Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION. Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Performed by the GEOS module (for geometry), geography is native Availability: 1.5 support for geography was introduced. For geography, this function has a distance tolerance of about 0.00001 meters and uses the sphere rather than spheroid calculation. NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 //s2.1.13.3 - ST\_Intersects(g1, g2) --> Not (ST\_Disjoint(g1, g2)) This method implements the SQL/MM specification. SQL-MM 3: 5.1.27 This method supports Circular Strings and Curves. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ST\_IsClosed** - Devuelve TRUE si los puntos de inicio y final de una LINESTRING son coincidentes. Para superficies poliedricas si son cerradas (volumetricas). Descripción Devuelve TRUE si los puntos de inicio y final de una LINESTRING son coincidentes. Para superficies poliédricas, te dice si las superficies son áreas (abiertas) o si son volumétricas (cerradas). This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 7.1.5, 9.3.3 SQL-MM define que el resultado de ST\_IsClosed(NULL) debe ser 0, mientras que PostGIS devuelve NULL. This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. Mejora: 2.0.0 se introdujo soporte de superficies poliédricas. This function supports Polyhedral surfaces.
- ST\_IsEmpty** - Tests if a geometry is empty. Descripción Devuelve True si la Geometría es una geometría vacía. Si es cierto, entonces esta Geometría representa una colección de geometrías vacías, polígonos vacíos, puntos vacíos, etc. SQL-MM define que el resultado de ST\_IsEmpty(NULL) debe ser 0, mientras que PostGIS devuelve NULL. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.7 This method supports Circular Strings and Curves. Cambiado: 2.0.0 En las versiones anteriores de PostGIS ST\_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') estaba permitido. Esto no esta permitido ahora en PostGIS 2.0.0 para ajustarse mejor a las normas SQL/MM.
- ST\_IsRing** - Tests if a LineString is closed and simple. Descripción Devuelve TRUE si esta LINESTRING es (ST\_StartPoint(g) ~= ST\_Endpoint(g)) y (no se intersecta con ella misma). This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1 This method implements the SQL/MM specification. SQL-MM 3: 7.1.6 SQL-MM define que el resultado de ST\_IsRing(NULL) debe ser 0, mientras que PostGIS devuelve NULL.
- ST\_IsSimple** - Devuelve (TRUE) si la geometría no tiene puntos geométricos anómalos, como auto intersecciones o tangencias. Descripción Devuelve TRUE si la geometría no tiene puntos geométricos anómalos, como auto intersecciones o tangencias. Para mas información sobre la definición del OGC de simplicidad y validez geométrica, visita el enlace "Ensuring OpenGIS compliancy of geometries" SQL-MM define que el resultado de ST\_IsSimple(NULL) debe ser 0, mientras que PostGIS devuelve NULL. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.8 This function supports 3d and will not drop the z-index.
- ST\_IsValid** - Tests if a geometry is well-formed in 2D. Descripción Tests if an ST\_Geometry value is well-formed and valid in 2D according to the OGC rules. For geometries with 3 and 4 dimensions, the validity is still only tested in 2 dimensions. For geometries that are invalid, a PostgreSQL NOTICE is emitted providing details of why it is not valid. For the version with the flags parameter, supported values are documented in This version does not print a NOTICE explaining invalidity. For more information on the definition of geometry validity, refer to SQL-MM defines the result of ST\_IsValid(NULL) to be 0, while PostGIS returns NULL. Performed by the GEOS module. The version accepting flags is available starting with 2.0.0. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 5.1.9 Neither OGC-SFS nor SQL-MM specifications include a flag argument for ST\_IsValid. The flag is a PostGIS extension.
- ST\_Length** - Returns the 2D length of a linear geometry. Descripción For geometry types: returns the 2D Cartesian length of the geometry if it is a LineString, MultiLineString, ST\_Curve, ST\_MultiCurve. For areal geometries 0 is returned; use instead. The units of length is determined by the spatial reference system of the geometry. For geography types: computation is performed using the inverse geodesic calculation. Units of length are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If use\_spheroid = false, then the calculation is

based on a sphere instead of a spheroid. Currently for geometry this is an alias for ST\_Length2D, but this may change to support higher dimensions. Changed: 2.0.0 Breaking change -- in prior versions applying this to a MULTI/POLYGON of type geography would give you the perimeter of the POLYGON/MULTIPOLYGON. In 2.0.0 this was changed to return 0 to be in line with geometry behavior. Please use ST\_Perimeter if you want the perimeter of a polygon For geography the calculation defaults to using a spheroidal model. To use the faster but less accurate spherical calculation use ST\_Length(gg,false); This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1 This method implements the SQL/MM specification. SQL-MM 3: 7.1.2, 9.3.4 Availability: 1.5.0 geography support was introduced in 1.5.

- **ST\_LineFromText** - Hace una geometría de la representación WKT con el SRID dado. Si SRID no se da, el valor predeterminado es 0. Descripción Hace una Geometry desde un WKT con el SRID dado. Si no se da un SRID, el valor predeterminado es 0. Si el WKT pasado no es un LINESTRING, se devuelve null. OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad Si sabes que todas tus geometrías son LINESTRING, es mas eficiente el uso de ST\_GeomFromText. Esto llama únicamente a ST\_GeomFromText y añade validaciones adicionales que devuelven un linestring. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 7.2.8
- **ST\_LineFromWKB** - Crea un LINESTRING desde un WKB con el SRID dado Descripción La función ST\_GeomFromWKB, toma una representación binaria "well-known" de una geometría y un ID de un Sistema de Referencia Espacial (SRID) y crea una instancia del tipo de geometría adecuado - en este caso una geometría LINESTRING. Esta función juega un rol de "Geometry Factory" en SQL. Si no se especifica un SRID, el valor predeterminado es 0. NULL se devuelve si la entrada bytea no representa un LINESTRING. OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad Si sabes que todas tus geometrías son LINESTRING, es mas eficiente el uso de . Esta función simplemente llama a y añade validaciones adicionales y devuelve una linestring. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 7.2.9
- **ST\_LinestringFromWKB** - Crea una geometría desde un WKB con el SRID dado. Descripción La función ST\_LinestringFromWKB, toma una representación de una geometría en "well-known binary" y un ID de un Sistema de Referencia Espacial (SRID) y crea una instancia del tipo apropiado de geometría - en este caso, una geometría LINESTRING. Esta función juega un rol de "Geometry Factory" en SQL. Si no se especifica un SRID, el valor predeterminado es 0.NULL se devuelve si la entrada bytea no representa una geometría LINESTRING. Esto es un alias para . OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad Si sabes que todas tus geometrías son LINESTRING, es mas eficiente el uso de . Esta función simplemente llama a y añade validaciones adicionales y devuelve una LINESTRING. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 7.2.9
- **ST\_LocateAlong** - Returns the point(s) on a geometry that match a measure value. Descripción Returns the location(s) along a measured geometry that have the given measure values. The result is a Point or MultiPoint. Polygonal inputs are not supported. If offset is provided, the result is offset to the left or right of the input line by the specified distance. A positive offset will be to the left, and a negative one to the right. Use this function only for linear geometries with an M component The semantic is specified by the ISO/IEC 13249-3 SQL/MM Spatial standard. Disponibilidad: 1.1.0 por antiguo nombre ST\_Locate\_Along\_Measure. Modificado: 2.0.0 en versiones anteriores éste solía llamarse ST\_Locate\_Along\_Measure. El nombre anterior ha quedado obsoleto y se eliminará en el futuro, pero aún está disponible. This function supports M coordinates. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.13
- **ST\_LocateBetween** - Returns the portions of a geometry that match a measure range. Descripción Devuelve un valor de la colección Geometry derivado con elementos que coinciden con la medida especificada. No se admiten elementos poligonales. Si se proporciona un desplazamiento, el resultado se desplazará a la izquierda o a la derecha de la línea de entrada por el número de unidades especificado. Un desplazamiento positivo será a la izquierda, y uno negativo a la derecha. Clipping a non-convex POLYGON may produce invalid geometry. The semantic is specified by the ISO/IEC 13249-3 SQL/MM Spatial standard. Disponibilidad: 1.1.0 por nombre antiguo ST\_Locate\_Between\_Measures. Modificado: 2.0.0 en versiones anteriores éste solía llamarse ST\_Locate\_Along\_Measure. El nombre anterior ha quedado obsoleto y se eliminará en el futuro, pero aún está disponible. Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. This function supports M coordinates. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- **ST\_M** - Returns the M coordinate of a Point. Descripción Devuelve la coordenada M del punto, o NULL si no seta disponible. La entrada debe ser un punto. Esto no es (todavía) parte de la especificación OGC, pero esta incluida aquí para completar la lista de extracción de coordenadas de un punto. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. This function supports 3d and will not drop the z-index.
- **ST\_MLineFromText** - Devuelve un valor especificado ST\_MultiLineString desde una representación WKT. Descripción Hace una Geometry desde el texto bien conocido (WKT) con el SRID dado. Si no se da un SRID, el valor predeterminado es

0. OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad Devuelve NULL si el WKT no es un MULTILINESTRING Si estas completamente seguro que todas tus geometrías WKT son puntos, no utilices esta función. Es mas lenta que ST\_GeomFromText ya que añade algunos pasos de validación. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 9.4.4
- **ST\_MPointFromText** - Hace una geometría desde un WKT con el SRID dado. Si no se da un SRID, el valor predeterminado es 0. Descripción Hace una geometría desde un WKT con el SRID dado. Si no se da un SRID, el valor predeterminado es 0. OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad Devuelve NULL si el WKT no es un MULTIPUNTO Si estas completamente seguro que todas tus geometrías WKT son puntos, no utilices esta función. Es mas lenta que ST\_GeomFromText ya que añade algunos pasos de validación. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 9.2.4
  - **ST\_MPolyFromText** - Hace una Geometría MultiPolygon desde un WKT con el SRID dado. Si no se da SRID, el valor predeterminado es 0. Descripción Hace un MultiPolygon desde un WKT con el SRID dado. Si no se da SRID, el valor predeterminado es 0. OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad Devuelve un error si el WKT no es un MULTIPOLYGON Si estas completamente seguro que todas tus geometrías WKT son multipolygon, no utilices esta función. Es mas lenta que ST\_GeomFromText ya que añade algunos pasos de validación adicionales. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 9.6.4
  - **ST\_ModEdgeHeal** - Cura dos aristas eliminando el nodo que las conecta, modificando la primera arista y eliminando la segunda arista. Devuelve el identificador del nodo eliminado. Descripción Cura dos aristas eliminando el nodo que las conecta, modificando la primera arista y eliminando la segunda arista. Devuelve el identificador del nodo eliminado. Actualiza todos los bordes y relaciones unidos existentes en consecuencia. Disponibilidad: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.9
  - **ST\_ModEdgeSplit** - Dividir un borde creando un nuevo nodo a lo largo de un borde existente, modificando el borde original y agregando un nuevo borde. Descripción Dividir un borde creando un nuevo nodo a lo largo de un borde existente, modificando el borde original y agregando un nuevo borde. Actualiza todos los bordes unidos existentes y relaciones en consecuencia. Devuelve el identificador del nodo recientemente agregado. Availability: 1.1 Cambiado: 2.0 - En versiones anteriores, esto fue mal llamado ST\_ModEdgesSplit This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.9
  - **ST\_MoveIsoNode** - Mueve un nodo aislado en una topología de un punto a otro. Si la nueva geometría apoint existe como nodo se lanza un error. Devuelve la descripción del movimiento. Descripción Mueve un nodo aislado en una topología de un punto a otro. Si la nueva geometría apoint existe como un nodo un error es lanzado. If any arguments are null, the apoint is not a point, the existing node is not isolated (is a start or end point of an existing edge), new node location intersects an existing edge (even at the end points) or the new location is in a different face (since 3.2.0) then an exception is thrown. Si el sistema de referencia espacial (SRID) de la geometría de punto no es el mismo que el de la topología se lanza una excepción. Disponibilidad: 2.0.0 Enhanced: 3.2.0 ensures the nod cannot be moved in a different face This method implements the SQL/MM specification. SQL-MM: Topo-Net Rutina: X.3.2
  - **ST\_NewEdgeHeal** - Cura dos aristas eliminando el nodo que las conecta, eliminando ambas aristas y sustituyéndolas por una arista cuya dirección sea la misma que la primera arista proporcionada. Descripción Cura dos aristas eliminando el nodo que las conecta, eliminando ambas aristas y sustituyéndolas por una arista cuya dirección sea la misma que la primera arista proporcionada. Devuelve el identificador de la nueva aristas reemplazante de las curadas. Actualiza todas las aristas y relaciones unidos existentes en consecuencia. Disponibilidad: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.9
  - **ST\_NewEdgesSplit** - Divide un borde creando un nuevo nodo a lo largo de un borde existente, eliminando el borde original y reemplazandolo con dos bordes nuevos. Devuelve el identificador del nuevo nodo creado que une los nuevos bordes. Descripción Divide un borde con el identificador de borde anedge creando un nodo nuevo con la localización del punto apoint a lo largo del borde actual, eliminando el borde original y reemplazando con dos bordes nuevos. Devuelve el identificador del nuevo nodo creado que une los nuevos bordes. Actualiza todos los bordes unidos existentes y relaciones en consecuencia. Si el sistema de referencia espacial (SRID) de la geometría de punto no es el mismo que el de la topología, el apoint no es una geometría de punto, el punto es nulo, el punto ya existe como un nodo, el borde no corresponde a un borde existente o el punto no está dentro del borde entonces se lanza una excepción. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Net Rutina: X.3.8



- **ST\_NumGeometries** - Devuelve el numero de puntos en la geometría. Funciona con todas las geometrías. Descripción Devuelve el numero de geometrías. Si la geometría es una GEOMETRYCOLLECTION (o MULTI\*) devuelve el numero de geometrías, para geometrías simples devuelve 1, si no devuelve NULL. Mejorado: 2.0.0 se introdujo soporte para superficies poliédricas, Triangulos y TIN. Cambiado: 2.0.0 En versiones anteriores esto devolvería NULL si la geometría no era de tipo collection/MULTI. 2.0.0+ devuelve 1 para geometrías simples, por ejemplo, POLYGON, LINESTRING, POINT. This method implements the SQL/MM specification. SQL-MM 3: 9.1.4 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- **ST\_NumInteriorRings** - Devuelva el número de anillos interiores de una geometría poligonal. Descripción Devuelve el número de anillos interiores de una geometría poligonal. Devuelve NULL si la geometría no es un polígono. This method implements the SQL/MM specification. SQL-MM 3: 8.2.5 Cambiado: 2.0.0 - En versiones anteriores permitiría pasar un multipolígono, devolviendo el número de anillos interiores de primer polígono.
- **ST\_NumPatches** - Devuelve el número de caras en una superficie poliédrica. Devolverá nulo para geometrías no poliédricas. Descripción Devuelve el número de caras en una superficie poliédrica. Devolverá nulo para geometrías no poliédricas. Esto es un alias para ST\_NumGeometries para admitir nombres MM. Más rápido para usar ST\_NumGeometries si no te importa la convención MM. Disponibilidad: 2.0.0 This function supports 3d and will not drop the z-index. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5 This function supports Polyhedral surfaces.
- **ST\_NumPoints** - Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString. Descripción Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString. Antes de 1.4 sólo funcionaba con cadenas de línea como el estado de especificaciones. A partir de 1.4, esto es un alias para ST\_NPoints que devuelve el número de vértices para no sólo las cadenas de línea. Considere el uso de ST\_NPoints en su lugar, que es multiuso y funciona con muchos tipos de geometría. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 7.2.4
- **ST\_OrderingEquals** - Tests if two geometries represent the same geometry and have points in the same directional order Description ST\_OrderingEquals compares two geometries and returns t (TRUE) if the geometries are equal and the coordinates are in the same order; otherwise it returns f (FALSE). This function is implemented as per the ArcSDE SQL specification rather than SQL-MM. [http://edndoc.esri.com/arcade/9.1/sql\\_api/sqlapi3.htm#ST\\_OrderingEquals](http://edndoc.esri.com/arcade/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals) This method implements the SQL/MM specification. SQL-MM 3: 5.1.43
- **ST\_Overlaps** - Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other Description Returns TRUE if geometry A and B "spatially overlap". Two geometries overlap if they have the same dimension, their interiors intersect in that dimension. and each has at least one point inside the other (or equivalently, neither one covers the other). The overlaps relation is symmetric and irreflexive. In mathematical terms:  $ST\_Overlaps(A, B) \Leftrightarrow (dim(A) = dim(B) = dim(Int(A) \cap Int(B)) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B))$  This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Overlaps`. Performed by the GEOS module Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.32
- **ST\_PatchN** - Devuelve el tipo de geometría del valor de ST\_Geometry. Descripción Devuelve la 1 geometría de base n-ésima (cara) si la geometría es un POLYHEDRALSURFACE, POLYHEDRALSURFACEM. De lo contrario, devuelve NULL. Esto devuelve la misma respuesta que ST\_GeometryN para las superficies de poliedros. Usar ST\_GeometryN es más rápido. El índice está basado en 1. Si desea extraer todas las geometrías, de una geometría, ST\_Dump es más eficiente. Disponibilidad: 2.0.0 This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces.
- **ST\_Perimeter** - Returns the length of the boundary of a polygonal geometry or geography. Descripción Returns the 2D perimeter of the geometry/geography if it is a ST\_Surface, ST\_MultiSurface (Polygon, MultiPolygon). 0 is returned for non-areal geometries. For linear geometries use . For geometry types, units for perimeter measures are specified by the spatial reference system of the geometry. For geography types, the calculations are performed using the inverse geodesic problem, where perimeter units are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If use\_spheroid = false, then calculations will approximate a sphere instead of a spheroid. Currently this is an alias for ST\_Perimeter2D, but this may change to support higher dimensions. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1 This method implements the SQL/MM specification. SQL-MM 3: 8.1.3, 9.5.4 Availability 2.0.0: Support for geography was introduced

- **ST\_Point** - Creates a Point with X, Y and SRID values. Descripción Returns a Point with the given X and Y coordinate values. This is the SQL-MM equivalent for that takes just X and Y. For geodetic coordinates, X is longitude and Y is latitude Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry. This method implements the SQL/MM specification. SQL-MM 3: 6.1.2
- **ST\_PointFromText** - Crea una geometría puntual desde un WKT con el SRID dado. Si no se especifica el SRID por defecto será unknown. Descripción Construye un objeto de punto de PostGIS ST\_GEOMETRY de la representación bien conocida del texto de OGC. Si no se da SRID, se omite a desconocido (actualmente 0). Si la geometría no es una representación de punto WKT, devuelve null. Si WKT es totalmente inválido, entonces lanza un error. Hay 2 variantes de la función ST\_PointFromText, la primera no toma SRID y devuelve una geometría sin sistema de referencia espacial definido. La segunda toma un id de un sistema de referencia como segundo argumento y devuelve una ST\_Geometry que incluye este srid como parte de sus metadatos. El srid debe estar definido en la tabla spatial\_ref\_sys. Si estas completamente seguro que todas tus geometrias WKT son puntos, no utilices esta función. Es mas lenta que ST\_GeomFromText ya que añade algunos pasos de validación. Si estas construyendo puntos desde coordenadas long lat y te interesan mas el rendimiento y la precisión que la conformidad con OGC, utiliza o el alias conforme al OGC. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - la opción SRID es de la suite de conformidad. This method implements the SQL/MM specification. SQL-MM 3: 6.1.8
- **ST\_PointFromWKB** - Crea una geometría desde un WKB con el SRID dado. Descripción La función ST\_PointFromWKB, toma una representación binaria "well-known" de una geometría y un ID de un Sistema de Referencia Espacial (SRID) y crea una instancia del tipo de geometría adecuado - en este caso una geometría POINT. Esta función juega un rol de "Geometry Factory" en SQL. Si no se especifica un SRID, el valor predeterminado es 0. NULL se devuelve si la entrada bytea no representa una geometría de POINT. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2 This method implements the SQL/MM specification. SQL-MM 3: 6.1.9 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves.
- **ST\_PointN** - Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString. Descripción Devuelve el punto enésimo en una sola cadena de línea o cadena de línea circular en la geometría. Los valores negativos se contabilizan hacia atrás desde el final de la cadena de línea, por lo que -1 es el último punto. Devuelve NULL si no hay cadena de línea en la geometría. El índice se basa en 1 como para las especificaciones OGC desde la versión 0.8.0. La indexación hacia atrás (índice negativo) no se encuentra en versiones anteriores de OGC implementado esto como basado en 0 en su lugar. Si desea obtener el punto n-ésimo de cada cadena de línea en una multiple cadena de línea, utilícelo en conjunción con ST\_Dump This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 7.2.5, 7.3.5 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. Cambiado: 2.0.0 ya no funciona con una sola geometría multilinestrings. En versiones antiguas de PostGIS -- una sola línea MultiLineString trabajaría felizmente con esta función y regresaría el punto de inicio. En 2.0.0 sólo devuelve NULL como cualquier otro MultiLineString. Cambiado: 2.3.0: indexación negativa disponible (-1 es el último punto)
- **ST\_PointOnSurface** - Computes a point guaranteed to lie in a polygon, or on a geometry. Descripción Returns a POINT which is guaranteed to lie in the interior of a surface (POLYGON, MULTIPOLYGON, and CURVED POLYGON). In PostGIS this function also works on line and point geometries. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.14.2 // s3.2.18.2 This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. The specifications define ST\_PointOnSurface for surface geometries only. PostGIS extends the function to support all common geometry types. Other databases (Oracle, DB2, ArcSDE) seem to support this function only for surfaces. SQL Server 2008 supports all common geometry types. This function supports 3d and will not drop the z-index.
- **ST\_Polygon** - Creates a Polygon from a LineString with a specified SRID. Descripción Returns a polygon built from the given LineString and sets the spatial reference system from the srid. ST\_Polygon is similar to Variant 1 with the addition of setting the SRID. , , Esta función no acepta una MULTILINESTRING. Utiliza o para generar una linestring. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 8.3.2 This function supports 3d and will not drop the z-index.
- **ST\_PolygonFromText** - Hace una geometría desde un WKT con el SRID dado. Si no se da un SRID, el valor predeterminado es 0. Descripción Hace una geometría desde WKT con el SRID dado. Si no se da SRID, el valor predeterminado es 0. Devuelve null si WKT no es un polígono. OGC SPEC 3.2.6.2 - La opción SRID es del paquete de conformidad Si estas completamente seguro que todas tus geometrias WKT son poligonos, no utilices esta función. Es mas lenta que ST\_GeomFromText ya que añade algunos pasos de validación adicionales. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 8.3.6

- ST\_Relate** - Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix Description These functions allow testing and evaluating the spatial (topological) relationship between two geometries, as defined by the Dimensionally Extended 9-Intersection Model (DE-9IM). The DE-9IM is specified as a 9-element matrix indicating the dimension of the intersections between the Interior, Boundary and Exterior of two geometries. It is represented by a 9-character text string using the symbols 'F', '0', '1', '2' (e.g. 'FF1FF0102'). A specific kind of spatial relationship can be tested by matching the intersection matrix to an intersection matrix pattern. Patterns can include the additional symbols 'T' (meaning "intersection is non-empty") and '\*' (meaning "any value"). Common spatial relationships are provided by the named functions , , , , , , , , and . Using an explicit pattern allows testing multiple conditions of intersects, crosses, etc in one step. It also allows testing spatial relationships which do not have a named spatial relationship function. For example, the relationship "Interior-Intersects" has the DE-9IM pattern T\*\*\*\*\*, which is not evaluated by any named predicate. For more information refer to . Variant 1: Tests if two geometries are spatially related according to the given intersectionMatrixPattern. Unlike most of the named spatial relationship predicates, this does NOT automatically include an index call. The reason is that some relationships are true for geometries which do NOT intersect (e.g. Disjoint). If you are using a relationship pattern that requires intersection, then include the && index call. It is better to use a named relationship function if available, since they automatically use a spatial index where one exists. Also, they may implement performance optimizations which are not available with full relate evaluation. Variant 2: Returns the DE-9IM matrix string for the spatial relationship between the two input geometries. The matrix string can be tested for matching a DE-9IM pattern using . Variant 3: Like variant 2, but allows specifying a Boundary Node Rule. A boundary node rule allows finer control over whether the endpoints of MultiLineStrings are considered to lie in the DE-9IM Interior or Boundary. The boundaryNodeRule values are: 1: OGC-Mod2 - line endpoints are in the Boundary if they occur an odd number of times. This is the rule defined by the OGC SFS standard, and is the default for ST\_Relate. 2: Endpoint - all endpoints are in the Boundary. 3: MultivalentEndpoint - endpoints are in the Boundary if they occur more than once. In other words, the boundary is all the "attached" or "inner" endpoints (but not the "unattached/outer" ones). 4: MonovalentEndpoint - endpoints are in the Boundary if they occur only once. In other words, the boundary is all the "unattached" or "outer" endpoints. This function is not in the OGC spec, but is implied. see s2.1.13.2 This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.25 Performed by the GEOS module Enhanced: 2.0.0 - added support for specifying boundary node rule. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION
- ST\_RemEdgeModFace** - Removes an edge, and if the edge separates two faces deletes one face and modifies the other face to cover the space of both. Descripción Removes an edge, and if the removed edge separates two faces deletes one face and modifies the other face to cover the space of both. Preferentially keeps the face on the right, to be consistent with . Returns the id of the face which is preserved. Actualiza todos los bordes unidos y relaciones en consecuencia existentes. Se niega a eliminar un borde que participa en la definición de un TopoGeometry existente. Se niega a sanear dos caras si cualquier TopoGeometry es definido por sólo uno de ellos (y no el otro). Si algún argumento es null, se desconoce el borde dado (debe existir ya en la tabla edge del esquema de topología), el nombre de la topología no es válido entonces se produce un error. Disponibilidad: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.15
- ST\_RemEdgeNewFace** - Elimina un borde y, si el borde eliminado separa dos caras, borra las caras originales y las reemplaza con una nueva cara. Descripción Elimina un borde y, si el borde eliminado separa dos caras, borra las caras originales y las reemplaza con una nueva cara. Devuelve el identificador de una cara creada recientemente o NULL, si no se crea ninguna nueva cara. No se crea ninguna nueva cara cuando el borde eliminado está colgando o aislado o confinado con la cara del universo (posiblemente haciendo que el universo se inunde en la cara del otro lado). Actualiza todos los bordes unidos y relaciones en consecuencia existentes. Se niega a eliminar un borde que participa en la definición de un TopoGeometry existente. Se niega a sanear dos caras si cualquier TopoGeometry es definido por sólo uno de ellos (y no el otro). Si algún argumento es null, se desconoce el borde dado (debe existir ya en la tabla edge del esquema de topología), el nombre de la topología no es válido entonces se produce un error. Disponibilidad: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X.3.14
- ST\_RemoveIsoEdge** - Elimina un borde aislado y devuelve la descripción de la acción. Si el borde no está aislado, se lanza una excepción. Descripción Elimina un borde aislado y devuelve la descripción de la acción. Si el borde no está aislado, se lanza una excepción. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X+1.3.3
- ST\_RemoveIsoNode** - Elimina un nodo aislado y devuelve la descripción de la acción. Si el nodo no está aislado (es el inicio o el final de un borde), entonces se lanza una excepción. Descripción Elimina un nodo aislado y devuelve la descripción de la acción. Si el nodo no está aislado (es el inicio o el final de un borde), entonces se lanza una excepción. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Geo y Topo-Net 3: Detalles de Rutina: X+1.3.3

- **ST\_SRID** - Returns the spatial reference identifier for a geometry. Description Returns the spatial reference identifier for the ST\_Geometry as defined in spatial\_ref\_sys table. spatial\_ref\_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.5 This method supports Circular Strings and Curves.
- **ST\_StartPoint** - Returns the first point of a LineString. Descripción Devuelve el primer punto de una geometría LINESTRING o CIRCULARLINESTRING como un POINT o NULL si el parámetro de entrada no es un LINESTRING o CIRCULARLINESTRING. This method implements the SQL/MM specification. SQL-MM 3: 7.1.3 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString. Cambiado: 2.0.0 ya no funciona con multilinestrings de geometrías simples. En versiones anteriores de PostGIS -- una línea simple multilinestring funciona sin problemas con esta función y devuelve el punto inicial. En la version 2.0.0 simplemente devuelve NULL como con cualquier multilinestring. La antigua version era una función sin documentar, pero la gente que asumía que tenia sus datos almacenados en LINESTRING pueden experimentar este comportamiento ahora de resultado NULL en la version 2.0.
- **ST\_SymDifference** - Computes a geometry representing the portions of geometries A and B that do not intersect. Description Returns a geometry representing the portions of geometries A and B that do not intersect. This is equivalent to ST\_Union(A,B) - ST\_Intersection(A,B). It is called a symmetric difference because  $ST\_SymDifference(A,B) = ST\_SymDifference(B,A)$ . If the optional gridSize argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher) Performed by the GEOS module Enhanced: 3.1.0 accept a gridSize parameter. Requires GEOS  $\geq 3.9.0$  to use the gridSize parameter This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.21 This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.
- **ST\_Touches** - Tests if two geometries have at least one point in common, but their interiors do not intersect Description Returns TRUE if A and B intersect, but their interiors do not intersect. Equivalently, A and B have at least one point in common, and the common points lie in at least one boundary. For Point/Point inputs the relationship is always FALSE, since points do not have a boundary. In mathematical terms:  $ST\_Touches(A, B) \Leftrightarrow (Int(A) \cap Int(B) \neq \emptyset) \wedge (A \cap B \neq \emptyset)$  This relationship holds if the DE-9IM Intersection Matrix for the two geometries matches one of: FT\*\*\*\*\* F\*\*T\*\*\*\*\* F\*\*\*T\*\*\*\* This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid using an index, use \_ST\_Touches instead. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.28
- **ST\_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system. Description Returns a new geometry with its coordinates transformed to a different spatial reference system. The destination spatial reference to\_srid may be identified by a valid SRID integer parameter (i.e. it must exist in the spatial\_ref\_sys table). Alternatively, a spatial reference defined as a PROJ.4 string can be used for to\_proj and/or from\_proj, however these methods are not optimized. If the destination spatial reference system is expressed with a PROJ.4 string instead of an SRID, the SRID of the output geometry will be set to zero. With the exception of functions with from\_proj, input geometries must have a defined SRID. ST\_Transform is often confused with . ST\_Transform actually changes the coordinates of a geometry from one spatial reference system to another, while ST\_SetSRID() simply changes the SRID identifier of the geometry. ST\_Transform automatically selects a suitable conversion pipeline given the source and target spatial reference systems. To use a specific conversion method, use . Requires PostGIS be compiled with PROJ support. Use to confirm you have PROJ support compiled in. If using more than one transformation, it is useful to have a functional index on the commonly used transformations to take advantage of index usage. Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+ Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Enhanced: 2.3.0 support for direct PROJ.4 text was introduced. This method implements the SQL/MM specification. SQL-MM 3: 5.1.6 This method supports Circular Strings and Curves. This function supports Polyhedral surfaces.
- **ST\_Union** - Computes a geometry representing the point-set union of the input geometries. Description Unions the input geometries, merging geometry to produce a result geometry with no overlaps. The output may be an atomic geometry, a MultiGeometry, or a Geometry Collection. Comes in several variants: Two-input variant: returns a geometry that is the union of two input geometries. If either input is NULL, then NULL is returned. Array variant: returns a geometry that is the union of an array of geometries. Aggregate variant: returns a geometry that is the union of a rowset of geometries. The ST\_Union() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of



data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries. See for a non-aggregate, single-input variant. The ST\_Union array and set variants use the fast Cascaded Union algorithm described in <http://blog.cleverelephant.ca/2009/01/must-faster-unions-in-postgis-14.html>. A gridSize can be specified to work in fixed-precision space. The inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher) may sometimes be used in place of ST\_Union, if the result is not required to be non-overlapping. ST\_Collect is usually faster than ST\_Union because it performs no processing on the collected geometries. Performed by the GEOS module. ST\_Union creates MultiLineString and does not sew LineStrings into a single LineString. Use to sew LineStrings. NOTE: this function was formerly called GeomUnion(), which was renamed from "Union" because UNION is an SQL reserved word. Enhanced: 3.1.0 accept a gridSize parameter. Requires GEOS >= 3.9.0 to use the gridSize parameter. Changed: 3.0.0 does not depend on SFCGAL. Availability: 1.4.0 - ST\_Union was enhanced. ST\_Union(geomarray) was introduced and also faster aggregate collection in PostgreSQL. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 Aggregate version is not explicitly defined in OGC SPEC. This method implements the SQL/MM specification. SQL-MM 3: 5.1.19 the z-index (elevation) when polygons are involved. This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

- **ST\_Volume** - Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0. Descripción Devuelve el volumen de un sólido 3D. Disponibilidad: 2.2.0 This method needs SFCGAL backend. This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN). This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 9.1 (same as ST\_3DVolume)
- **ST\_WKBToSQL** - Devuelve un valor específico de ST\_Geometry desde una representación "Well-Known Binary" (WKB). Es un alias para ST\_GeomFromWKB que no toma srid Descripción This method implements the SQL/MM specification. SQL-MM 3: 5.1.36
- **ST\_WKTTToSQL** - Devuelve un valor específico de ST\_Geometry desde una representación "Well-Known Text" (WKT). Es un alias para ST\_GeomFromText Descripción This method implements the SQL/MM specification. SQL-MM 3: 5.1.34
- **ST\_Within** - Tests if every point of A lies in B, and their interiors have a point in common Description Returns TRUE if geometry A is within geometry B. A is within B if and only if all points of A lie inside (i.e. in the interior or boundary of) B (or equivalently, no points of A lie in the exterior of B), and the interiors of A and B have at least one point in common. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID. In mathematical terms:  $ST\_Within(A, B) \Leftrightarrow (A \cap B = A) \wedge (Int(A) \cap Int(B) \neq \emptyset)$  The within relation is reflexive: every geometry is within itself. The relation is antisymmetric: if  $ST\_Within(A,B) = true$  and  $ST\_Within(B,A) = true$ , then the two geometries must be topologically equal ( $ST\_Equals(A,B) = true$ ). ST\_Within is the converse of . So,  $ST\_Within(A,B) = ST\_Contains(B,A)$ . Because the interiors must have a common point, a subtlety of the definition is that lines and points lying fully in the boundary of polygons or lines are not within the geometry. For further details see Subtleties of OGC Covers, Contains, Within. The predicate provides a more inclusive relationship. This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function ST\_Within. Performed by the GEOS module Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Do not use this function with invalid geometries. You will get unexpected results. NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - a.Relate(b, 'T\*\*F\*\*F\*\*\*') This method implements the SQL/MM specification. SQL-MM 3: 5.1.30
- **ST\_X** - Returns the X coordinate of a Point. Descripción Devuelve la coordenada X del punto, o NULL si no está disponible. La entrada debe ser un punto. To get the minimum and maximum X value of geometry coordinates use the functions ST\_MinX and ST\_MaxX. This method implements the SQL/MM specification. SQL-MM 3: 6.1.3 This function supports 3d and will not drop the z-index.
- **ST\_Y** - Returns the Y coordinate of a Point. Descripción Devuelve la coordenada Y del punto, o NULL si no está disponible. La entrada debe ser un punto. To get the minimum and maximum Y value of geometry coordinates use the functions ST\_MinY and ST\_MaxY. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 6.1.4 This function supports 3d and will not drop the z-index.
- **ST\_Z** - Returns the Z coordinate of a Point. Descripción Devuelve la coordenada Z del punto, o NULL si no está disponible. La entrada debe ser un punto. To get the minimum and maximum Z value of geometry coordinates use the functions ST\_MinZ and ST\_MaxZ. This method implements the SQL/MM specification. This function supports 3d and will not drop the z-index.

- **TG\_ST\_SRID** - Returns the spatial reference identifier for a topogeometry. Descripción Returns the spatial reference identifier for the ST\_Geometry as defined in spatial\_ref\_sys table. spatial\_ref\_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries. Availability: 3.2.0 This method implements the SQL/MM specification. SQL-MM 3: 14.1.5

## 12.4 PostGIS Geography Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a **geography** data type object.



### Note

Functions with a (T) are not native geodetic functions, and use a ST\_Transform call to and from geometry to do the operation. As a result, they may not behave as expected when going over dateline, poles, and for large geometries or geometry pairs that cover more than one UTM zone. Basic transform - (favoring UTM, Lambert Azimuthal (North/South), and falling back on mercator in worst case scenario)

- **ST\_Area** - Returns the area of a polygonal geometry.
- **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsEWKT** - Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.
- **ST\_AsGML** - Return the geometry as a GML version 2 or 3 element.
- **ST\_AsGeoJSON** - Return a geometry as a GeoJSON element.
- **ST\_AsKML** - Return the geometry as a KML element.
- **ST\_AsSVG** - Returns SVG path data for a geometry.
- **ST\_AsText** - Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.
- **ST\_Azimuth** - Returns the north-based azimuth of a line between two points.
- **ST\_Buffer** - Computes a geometry covering all points within a given distance from a geometry.
- **ST\_Centroid** - Returns the geometric center of a geometry.
- **ST\_ClosestPoint** - Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.
- **ST\_CoveredBy** - Tests if every point of A lies in B
- **ST\_Covers** - Tests if every point of B lies in A
- **ST\_DWithin** - Tests if two geometries are within a given distance
- **ST\_Distance** - Returns the distance between two geometry or geography values.
- **ST\_GeogFromText** - Devuelve un valor específico "geography" desde una representación "Well-Known Text" (WKT) o extendida.
- **ST\_GeogFromWKB** - Crea una instancia "geography" desde la representación de una geometría en "Well-Known Binary" (WKB) o "Extended Well-Known Binary" (EWKB).
- **ST\_GeographyFromText** - Devuelve un valor específico "geography" desde una representación "Well-Known Text" (WKT) o extendida.

- **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **ST\_Intersection** - Computes a geometry representing the shared portion of geometries A and B.
- **ST\_Intersects** - Tests if two geometries intersect (they have at least one point in common)
- **ST\_Length** - Returns the 2D length of a linear geometry.
- **ST\_LineInterpolatePoint** - Returns a point interpolated along a line at a fractional location.
- **ST\_LineInterpolatePoints** - Returns points interpolated along a line at a fractional interval.
- **ST\_LineLocatePoint** - Returns the fractional location of the closest point on a line to a point.
- **ST\_LineSubstring** - Returns the part of a line between two fractional locations.
- **ST\_Perimeter** - Returns the length of the boundary of a polygonal geometry or geography.
- **ST\_Project** - Returns a point projected from a start point by a distance and bearing (azimuth).
- **ST\_Segmentize** - Returns a modified geometry/geography having no segment longer than a given distance.
- **ST\_ShortestLine** - Returns the 2D shortest line between two geometries
- **ST\_Summary** - Devuelve un resumen de texto del contenido de la geometría.
- **<->** - Returns the 2D distance between A and B.
- **&&** - Returns TRUE if A's 2D bounding box intersects B's 2D bounding box.

## 12.5 PostGIS Raster Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a **raster** data type object. Listed in alphabetical order.

- **Box3D** - Returns the box 3d representation of the enclosing box of the raster.
- **@** - Returns TRUE if A's bounding box is contained by B's. Uses double precision bounding box.
- **~** - Returns TRUE if A's bounding box is contains B's. Uses double precision bounding box.
- **=** - Returns TRUE if A's bounding box is the same as B's. Uses double precision bounding box.
- **&&** - Returns TRUE if A's bounding box intersects B's bounding box.
- **&<** - Returns TRUE if A's bounding box is to the left of B's.
- **&>** - Returns TRUE if A's bounding box is to the right of B's.
- **~=** - Returns TRUE if A's bounding box is the same as B's.
- **ST\_Retile** - Return a set of configured tiles from an arbitrarily tiled raster coverage.
- **ST\_AddBand** - Returns a raster with the new band(s) of given type added with given initial value in the given index location. If no index is specified, the band is added to the end.
- **ST\_AsBinary/ST\_AsWKB** - Return the Well-Known Binary (WKB) representation of the raster.
- **ST\_AsGDALRaster** - Return the raster tile in the designated GDAL Raster format. Raster formats are one of those supported by your compiled library. Use ST\_GDALDrivers() to get a list of formats supported by your library.
- **ST\_AsHexWKB** - Return the Well-Known Binary (WKB) in Hex representation of the raster.

- **ST\_AsJPEG** - Return the raster tile selected bands as a single Joint Photographic Exports Group (JPEG) image (byte array). If no band is specified and 1 or more than 3 bands, then only the first band is used. If only 3 bands then all 3 bands are used and mapped to RGB.
  - **ST\_AsPNG** - Return the raster tile selected bands as a single portable network graphics (PNG) image (byte array). If 1, 3, or 4 bands in raster and no bands are specified, then all bands are used. If more 2 or more than 4 bands and no bands specified, then only band 1 is used. Bands are mapped to RGB or RGBA space.
  - **ST\_AsRaster** - Converts a PostGIS geometry to a PostGIS raster.
  - **ST\_AsTIFF** - Return the raster selected bands as a single TIFF image (byte array). If no band is specified or any of specified bands does not exist in the raster, then will try to use all bands.
  - **ST\_Aspect** - Returns the aspect (in degrees by default) of an elevation raster band. Useful for analyzing terrain.
  - **ST\_Band** - Returns one or more bands of an existing raster as a new raster. Useful for building new rasters from existing rasters.
  - **ST\_BandFileSize** - Returns the file size of a band stored in file system. If no bandnum specified, 1 is assumed.
  - **ST\_BandFileTimestamp** - Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.
  - **ST\_BandIsNoData** - Returns true if the band is filled with only nodata values.
  - **ST\_BandMetaData** - Returns basic meta data for a specific raster band. band num 1 is assumed if none-specified.
  - **ST\_BandNoDataValue** - Returns the value in a given band that represents no data. If no band num 1 is assumed.
  - **ST\_BandPath** - Returns system file path to a band stored in file system. If no bandnum specified, 1 is assumed.
  - **ST\_BandPixelType** - Returns the type of pixel for given band. If no bandnum specified, 1 is assumed.
  - **ST\_Clip** - Returns the raster clipped by the input geometry. If band number not is specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped.
  - **ST\_ColorMap** - Creates a new raster of up to four 8BUI bands (grayscale, RGB, RGBA) from the source raster and a specified band. Band 1 is assumed if not specified.
  - **ST\_Contains** - Return true if no points of raster rastB lie in the exterior of raster rastA and at least one point of the interior of rastB lies in the interior of rastA.
  - **ST\_ContainsProperly** - Return true if rastB intersects the interior of rastA but not the boundary or exterior of rastA.
  - **ST\_Contour** - Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.
  - **ST\_ConvexHull** - Return the convex hull geometry of the raster including pixel values equal to BandNoDataValue. For regular shaped and non-skewed rasters, this gives the same result as ST\_Envelope so only useful for irregularly shaped or skewed rasters.
  - **ST\_Count** - Returns the number of pixels in a given band of a raster or raster coverage. If no band is specified defaults to band 1. If exclude\_nodata\_value is set to true, will only count pixels that are not equal to the nodata value.
  - **ST\_CountAgg** - Aggregate. Returns the number of pixels in a given band of a set of rasters. If no band is specified defaults to band 1. If exclude\_nodata\_value is set to true, will only count pixels that are not equal to the NODATA value.
  - **ST\_CoveredBy** - Return true if no points of raster rastA lie outside raster rastB.
  - **ST\_Covers** - Return true if no points of raster rastB lie outside raster rastA.
  - **ST\_DFullyWithin** - Return true if rasters rastA and rastB are fully within the specified distance of each other.
  - **ST\_DWithin** - Return true if rasters rastA and rastB are within the specified distance of each other.
  - **ST\_Disjoint** - Return true if raster rastA does not spatially intersect rastB.
-

- **ST\_DumpAsPolygons** - Returns a set of geomval (geom,val) rows, from a given raster band. If no band number is specified, band num defaults to 1.
- **ST\_DumpValues** - Get the values of the specified band as a 2-dimension array.
- **ST\_Envelope** - Returns the polygon representation of the extent of the raster.
- **ST\_FromGDALRaster** - Returns a raster from a supported GDAL raster file.
- **ST\_GeoReference** - Returns the georeference meta data in GDAL or ESRI format as commonly seen in a world file. Default is GDAL.
- **ST\_Grayscale** - Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue
- **ST\_HasNoBand** - Returns true if there is no band with given band number. If no band number is specified, then band number 1 is assumed.
- **ST\_Height** - Returns the height of the raster in pixels.
- **ST\_HillShade** - Returns the hypothetical illumination of an elevation raster band using provided azimuth, altitude, brightness and scale inputs.
- **ST\_Histogram** - Returns a set of record summarizing a raster or raster coverage data distribution separate bin ranges. Number of bins are autocomputed if not specified.
- **ST\_InterpolateRaster** - Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation.
- **ST\_Intersection** - Returns a raster or a set of geometry-pixelvalue pairs representing the shared portion of two rasters or the geometrical intersection of a vectorization of the raster and a geometry.
- **ST\_Intersects** - Return true if raster rastA spatially intersects raster rastB.
- **ST\_IsEmpty** - Returns true if the raster is empty (width = 0 and height = 0). Otherwise, returns false.
- **ST\_MakeEmptyCoverage** - Cover georeferenced area with a grid of empty raster tiles.
- **ST\_MakeEmptyRaster** - Returns an empty raster (having no bands) of given dimensions (width & height), upperleft X and Y, pixel size and rotation (scalex, scaley, skewx & skewy) and reference system (srid). If a raster is passed in, returns a new raster with the same size, alignment and SRID. If srid is left out, the spatial ref is set to unknown (0).
- **ST\_MapAlgebra (callback function version)** - Callback function version - Returns a one-band raster given one or more input rasters, band indexes and one user-specified callback function.
- **ST\_MapAlgebraExpr** - 1 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the input raster band and of pixeltype provided. Band 1 is assumed if no band is specified.
- **ST\_MapAlgebraExpr** - 2 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the two input raster bands and of pixeltype provided. band 1 of each raster is assumed if no band numbers are specified. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster and have its extent defined by the "extenttype" parameter. Values for "extenttype" can be: INTERSECTION, UNION, FIRST, SECOND.
- **ST\_MapAlgebraFct** - 1 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the input raster band and of pixeltype provided. Band 1 is assumed if no band is specified.
- **ST\_MapAlgebraFct** - 2 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the 2 input raster bands and of pixeltype provided. Band 1 is assumed if no band is specified. Extent type defaults to INTERSECTION if not specified.
- **ST\_MapAlgebraFctNgb** - 1-band version: Map Algebra Nearest Neighbor using user-defined PostgreSQL function. Return a raster which values are the result of a PLPGSQL user function involving a neighborhood of values from the input raster band.
- **ST\_MapAlgebra (expression version)** - Expression version - Returns a one-band raster given one or two input rasters, band indexes and one or more user-specified SQL expressions.

- **ST\_MemSize** - Returns the amount of space (in bytes) the raster takes.
  - **ST\_MetaData** - Returns basic meta data about a raster object such as pixel size, rotation (skew), upper, lower left, etc.
  - **ST\_MinConvexHull** - Return the convex hull geometry of the raster excluding NODATA pixels.
  - **ST\_NearestValue** - Returns the nearest non-NODATA value of a given band's pixel specified by a columnx and rowy or a geometric point expressed in the same spatial reference coordinate system as the raster.
  - **ST\_Neighborhood** - Returns a 2-D double precision array of the non-NODATA values around a given band's pixel specified by either a columnX and rowY or a geometric point expressed in the same spatial reference coordinate system as the raster.
  - **ST\_NotSameAlignmentReason** - Returns text stating if rasters are aligned and if not aligned, a reason why.
  - **ST\_NumBands** - Returns the number of bands in the raster object.
  - **ST\_Overlaps** - Return true if raster rastA and rastB intersect but one does not completely contain the other.
  - **ST\_PixelAsCentroid** - Returns the centroid (point geometry) of the area represented by a pixel.
  - **ST\_PixelAsCentroids** - Returns the centroid (point geometry) for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The point geometry is the centroid of the area represented by a pixel.
  - **ST\_PixelAsPoint** - Returns a point geometry of the pixel's upper-left corner.
  - **ST\_PixelAsPoints** - Returns a point geometry for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The coordinates of the point geometry are of the pixel's upper-left corner.
  - **ST\_PixelAsPolygon** - Returns the polygon geometry that bounds the pixel for a particular row and column.
  - **ST\_PixelAsPolygons** - Returns the polygon geometry that bounds every pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel.
  - **ST\_PixelHeight** - Returns the pixel height in geometric units of the spatial reference system.
  - **ST\_PixelOfValue** - Get the columnx, rowy coordinates of the pixel whose value equals the search value.
  - **ST\_PixelWidth** - Returns the pixel width in geometric units of the spatial reference system.
  - **ST\_Polygon** - Returns a multipolygon geometry formed by the union of pixels that have a pixel value that is not no data value. If no band number is specified, band num defaults to 1.
  - **ST\_Quantile** - Compute quantiles for a raster or raster table coverage in the context of the sample or population. Thus, a value could be examined to be at the raster's 25%, 50%, 75% percentile.
  - **ST\_RastFromHexWKB** - Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.
  - **ST\_RastFromWKB** - Return a raster value from a Well-Known Binary (WKB) raster.
  - **ST\_RasterToWorldCoord** - Returns the raster's upper left corner as geometric X and Y (longitude and latitude) given a column and row. Column and row starts at 1.
  - **ST\_RasterToWorldCoordX** - Returns the geometric X coordinate upper left of a raster, column and row. Numbering of columns and rows starts at 1.
  - **ST\_RasterToWorldCoordY** - Returns the geometric Y coordinate upper left corner of a raster, column and row. Numbering of columns and rows starts at 1.
  - **ST\_Reclass** - Creates a new raster composed of band types reclassified from original. The nband is the band to be changed. If nband is not specified assumed to be 1. All other bands are returned unchanged. Use case: convert a 16BUI band to a 8BUI and so forth for simpler rendering as viewable formats.
  - **ST\_Resample** - Resample a raster using a specified resampling algorithm, new dimensions, an arbitrary grid corner and a set of raster georeferencing attributes defined or borrowed from another raster.
-



- **ST\_Rescale** - Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline, Lanczos, Max or Min resampling algorithm. Default is NearestNeighbor.
  - **ST\_Resize** - Resize a raster to a new width/height
  - **ST\_Reskew** - Resample a raster by adjusting only its skew (or rotation parameters). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.
  - **ST\_Rotation** - Returns the rotation of the raster in radian.
  - **ST\_Roughness** - Returns a raster with the calculated "roughness" of a DEM.
  - **ST\_SRID** - Returns the spatial reference identifier of the raster as defined in spatial\_ref\_sys table.
  - **ST\_SameAlignment** - Returns true if rasters have same skew, scale, spatial ref, and offset (pixels can be put on same grid without cutting into pixels) and false if they don't with notice detailing issue.
  - **ST\_ScaleX** - Returns the X component of the pixel width in units of coordinate reference system.
  - **ST\_ScaleY** - Returns the Y component of the pixel height in units of coordinate reference system.
  - **ST\_SetBandIndex** - Update the external band number of an out-db band
  - **ST\_SetBandIsNoData** - Sets the isnodata flag of the band to TRUE.
  - **ST\_SetBandNoDataValue** - Sets the value for the given band that represents no data. Band 1 is assumed if no band is specified. To mark a band as having no nodata value, set the nodata value = NULL.
  - **ST\_SetBandPath** - Update the external path and band number of an out-db band
  - **ST\_SetGeoReference** - Set Georeference 6 georeference parameters in a single call. Numbers should be separated by white space. Accepts inputs in GDAL or ESRI format. Default is GDAL.
  - **ST\_SetM** - Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the M dimension using the requested resample algorithm.
  - **ST\_SetRotation** - Set the rotation of the raster in radian.
  - **ST\_SetSRID** - Sets the SRID of a raster to a particular integer srid defined in the spatial\_ref\_sys table.
  - **ST\_SetScale** - Sets the X and Y size of pixels in units of coordinate reference system. Number units/pixel width/height.
  - **ST\_SetSkew** - Sets the georeference X and Y skew (or rotation parameter). If only one is passed in, sets X and Y to the same value.
  - **ST\_SetUpperLeft** - Sets the value of the upper left corner of the pixel of the raster to projected X and Y coordinates.
  - **ST\_SetValue** - Returns modified raster resulting from setting the value of a given band in a given columnx, rowy pixel or the pixels that intersect a particular geometry. Band numbers start at 1 and assumed to be 1 if not specified.
  - **ST\_SetValues** - Returns modified raster resulting from setting the values of a given band.
  - **ST\_SetZ** - Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimension using the requested resample algorithm.
  - **ST\_SkewX** - Returns the georeference X skew (or rotation parameter).
  - **ST\_SkewY** - Returns the georeference Y skew (or rotation parameter).
  - **ST\_Slope** - Returns the slope (in degrees by default) of an elevation raster band. Useful for analyzing terrain.
  - **ST\_SnapToGrid** - Resample a raster by snapping it to a grid. New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.
-

- **ST\_Summary** - Returns a text summary of the contents of the raster.
- **ST\_SummaryStats** - Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed is no band is specified.
- **ST\_SummaryStatsAgg** - Aggregate. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a set of raster. Band 1 is assumed is no band is specified.
- **ST\_TPI** - Returns a raster with the calculated Topographic Position Index.
- **ST\_TRI** - Returns a raster with the calculated Terrain Ruggedness Index.
- **ST\_Tile** - Returns a set of rasters resulting from the split of the input raster based upon the desired dimensions of the output rasters.
- **ST\_Touches** - Return true if raster rastA and rastB have at least one point in common but their interiors do not intersect.
- **ST\_Transform** - Reprojects a raster in a known spatial reference system to another known spatial reference system using specified resampling algorithm. Options are NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos defaulting to NearestNeighbor.
- **ST\_Union** - Returns the union of a set of raster tiles into a single raster composed of 1 or more bands.
- **ST\_UpperLeftX** - Returns the upper left X coordinate of raster in projected spatial ref.
- **ST\_UpperLeftY** - Returns the upper left Y coordinate of raster in projected spatial ref.
- **ST\_Value** - Returns the value of a given band in a given columnx, rowy pixel or at a particular geometric point. Band numbers start at 1 and assumed to be 1 if not specified. If exclude\_nodata\_value is set to false, then all pixels include nodata pixels are considered to intersect and return value. If exclude\_nodata\_value is not passed in then reads it from metadata of raster.
- **ST\_ValueCount** - Returns a set of records containing a pixel band value and count of the number of pixels in a given band of a raster (or a raster coverage) that have a given set of values. If no band is specified defaults to band 1. By default nodata value pixels are not counted, and all other values in the pixel are output and pixel band values are rounded to the nearest integer.
- **ST\_Width** - Returns the width of the raster in pixels.
- **ST\_Within** - Return true if no points of raster rastA lie in the exterior of raster rastB and at least one point of the interior of rastA lies in the interior of rastB.
- **ST\_WorldToRasterCoord** - Returns the upper left corner as column and row given geometric X and Y (longitude and latitude) or a point geometry expressed in the spatial reference coordinate system of the raster.
- **ST\_WorldToRasterCoordX** - Returns the column in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw) represented in world spatial reference system of raster.
- **ST\_WorldToRasterCoordY** - Returns the row in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw) represented in world spatial reference system of raster.
- **UpdateRasterSRID** - Change the SRID of all rasters in the user-specified column and table.

## 12.6 PostGIS Geometry / Geography / Raster Dump Functions

The functions given below are PostGIS functions that take as input or return as output a set of or single **geometry\_dump** or **geomval** data type object.

- **ST\_DumpAsPolygons** - Returns a set of geomval (geom,val) rows, from a given raster band. If no band number is specified, band num defaults to 1.
- **ST\_Intersection** - Returns a raster or a set of geometry-pixelvalue pairs representing the shared portion of two rasters or the geometrical intersection of a vectorization of the raster and a geometry.



## 12.7 PostGIS Box Functions

The functions given below are PostGIS functions that take as input or return as output the box\* family of PostGIS spatial types. The box family of types consists of **box2d**, and **box3d**

- **Box2D** - Returns a BOX2D representing the 2D extent of a geometry.
- **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
- **Box3D** - Returns the box 3d representation of the enclosing box of the raster.
- **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST\_3DMakeBox** - Creates a BOX3D defined by two 3D point geometries.
- **ST\_AsMVTGeom** - Transforms a geometry into the coordinate space of a MVT tile.
- **ST\_AsTWKB** - Returns the geometry as TWKB, aka "Tiny Well-Known Binary"
- **ST\_Box2dFromGeoHash** - Devuelve un BOX2D de una cadena de GeoHash.
- **ST\_ClipByBox2D** - Computes the portion of a geometry falling within a rectangle.
- **ST\_EstimatedExtent** - Returns the estimated extent of a spatial table.
- **ST\_Expand** - Returns a bounding box expanded from another bounding box or a geometry.
- **ST\_Extent** - Aggregate function that returns the bounding box of geometries.
- **ST\_MakeBox2D** - Creates a BOX2D defined by two 2D point geometries.
- **ST\_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- **ST\_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST\_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- **ST\_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- **RemoveUnusedPrimitives** - Removes topology primitives which not needed to define existing TopoGeometry objects.
- **ValidateTopology** - Devuelve un conjunto de objetos validate\_topology\_return\_type que detallan problemas con la topología.
- **~(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (BOX2DF).
- **@(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- **@(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- **@(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- **&&(box2df,box2df)** - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- **&&(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- **&&(geometry,box2df)** - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

## 12.8 PostGIS Functions that support 3D

The functions given below are PostGIS functions that do not throw away the Z-Index.

- **AddGeometryColumn** - Suprime una columna de geometrías de una tabla espacial.
  - **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
  - **DropGeometryColumn** - Suprime una columna de geometrías de una tabla espacial.
  - **GeometryType** - Devuelve el tipo de geometría del valor de ST\_Geometry.
  - **ST\_3DArea** - Computes area of 3D surface geometries. Will return 0 for solids.
  - **ST\_3DClosestPoint** - Returns the 3D point on g1 that is closest to g2. This is the first point of the 3D shortest line.
  - **ST\_3DConvexHull** - Computes the 3D convex hull of a geometry.
  - **ST\_3DDFullyWithin** - Tests if two 3D geometries are entirely within a given 3D distance
  - **ST\_3DDWithin** - Tests if two 3D geometries are within a given 3D distance
  - **ST\_3DDifference** - Perform 3D difference
  - **ST\_3DDistance** - Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units.
  - **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
  - **ST\_3DIntersection** - Perform 3D intersection
  - **ST\_3DIntersects** - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
  - **ST\_3DLength** - Returns the 3D length of a linear geometry.
  - **ST\_3DLineInterpolatePoint** - Returns a point interpolated along a 3D line at a fractional location.
  - **ST\_3DLongestLine** - Returns the 3D longest line between two geometries
  - **ST\_3DMaxDistance** - Returns the 3D cartesian maximum distance (based on spatial ref) between two geometries in projected units.
  - **ST\_3DPerimeter** - Returns the 3D perimeter of a polygonal geometry.
  - **ST\_3DShortestLine** - Returns the 3D shortest line between two geometries
  - **ST\_3DUnion** - Perform 3D union.
  - **ST\_AddMeasure** - Interpolates measures along a linear geometry.
  - **ST\_AddPoint** - Añade un punto a una cadena de línea.
  - **ST\_Affine** - Apply a 3D affine transformation to a geometry.
  - **ST\_ApproximateMedialAxis** - Compute the approximate medial axis of an areal geometry.
  - **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
  - **ST\_AsEWKB** - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
  - **ST\_AsEWKT** - Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.
  - **ST\_AsGML** - Return the geometry as a GML version 2 or 3 element.
  - **ST\_AsGeoJSON** - Return a geometry as a GeoJSON element.
-

- **ST\_AsHEXEWKB** - Returns a Geometry in HEXEWKB format (as text) using either little-endian (NDR) or big-endian (XDR) encoding.
  - **ST\_AsKML** - Return the geometry as a KML element.
  - **ST\_AsX3D** - Returns a Geometry in X3D xml node element format: ISO-IEC-19776-1.2-X3DEncodings-XML
  - **ST\_Boundary** - Devuelve el cierre del limite combinatorio de esta geometría.
  - **ST\_BoundingDiagonal** - Devuelve la diagonal del cuadro delimitador de la geometría suministrada.
  - **ST\_CPAWithin** - Tests if the closest point of approach of two trajectories is within the specified distance.
  - **ST\_ChaikinSmoothing** - Returns a smoothed version of a geometry, using the Chaikin algorithm
  - **ST\_ClosestPointOfApproach** - Returns a measure at the closest point of approach of two trajectories.
  - **ST\_Collect** - Creates a GeometryCollection or Multi\* geometry from a set of geometries.
  - **ST\_ConstrainedDelaunayTriangles** - Return a constrained Delaunay triangulation around the given input geometry.
  - **ST\_ConvexHull** - Computes the convex hull of a geometry.
  - **ST\_CoordDim** - Devuelve la dimensión de las coordenadas del valor de ST\_Geometry.
  - **ST\_CurveToLine** - Converts a geometry containing curves to a linear geometry.
  - **ST\_DelaunayTriangles** - Returns the Delaunay triangulation of the vertices of a geometry.
  - **ST\_Difference** - Computes a geometry representing the part of geometry A that does not intersect geometry B.
  - **ST\_DistanceCPA** - Returns the distance between the closest point of approach of two trajectories.
  - **ST\_Dump** - Returns a set of geometry\_dump rows for the components of a geometry.
  - **ST\_DumpPoints** - Devuelve un resumen de texto del contenido de la geometría.
  - **ST\_DumpRings** - Returns a set of geometry\_dump rows for the exterior and interior rings of a Polygon.
  - **ST\_DumpSegments** - Devuelve un resumen de texto del contenido de la geometría.
  - **ST\_EndPoint** - Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString.
  - **ST\_ExteriorRing** - Devuelva el número de anillos interiores de una geometría poligonal.
  - **ST\_Extrude** - Extrude a surface to a related volume
  - **ST\_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
  - **ST\_Force2D** - Forzar las geometrías en un "modo de 2 dimensiones".
  - **ST\_ForceCurve** - Relanzar una geometría en su tipo curvo, si corresponde.
  - **ST\_ForceLHR** - Force LHR orientation
  - **ST\_ForcePolygonCCW** - Orienta todos los aros exteriores en sentido contrario a las agujas del reloj y todos los aros interiores en sentido horario.
  - **ST\_ForcePolygonCW** - Orienta todos los anillos exteriores en el sentido de las agujas del reloj y todos los anillos interiores en sentido contrario a las agujas del reloj.
  - **ST\_ForceRHR** - Fuerza la orientación de los vértices en un polígono para seguir la regla de la mano derecha.
  - **ST\_ForceSFS** - Fuerza las geometrías para usar sólo los tipos de geometría SFS 1.1.
  - **ST\_Force\_3D** - Forzar las geometrías en modo XYZ. Este es un alias para ST\_Force3DZ.
  - **ST\_Force\_3DZ** - Fuerza las geometrías en modo XYZ.
-

- **ST\_Force\_4D** - Fuerza las geometrías en modo XYZM.
  - **ST\_Force\_Collection** - Convertir la geometría en una GEOMETRYCOLLECTION.
  - **ST\_GeomFromEWKB** - Devuelve un valor específico de ST\_Geometry desde una representación "Extended Well-Known Binary" (EWKB).
  - **ST\_GeomFromEWKT** - Devuelve un valor especificado ST\_Geometry desde una representación "Extended Well-Known Text" (EWKT).
  - **ST\_GeomFromGML** - Toma una representación GML como entrada de una geometría y extrae un objeto geométrico PostGIS
  - **ST\_GeomFromGeoJSON** - Toma como entrada una representación geojson de una geometría y devuelve un objeto geométrico PostGIS
  - **ST\_GeomFromKML** - Toma una representación de una geometría KML de entrada y devuelve un objeto geométrico PostGIS
  - **ST\_GeometricMedian** - Returns the geometric median of a MultiPoint.
  - **ST\_GeometryN** - Devuelve el tipo de geometría del valor de ST\_Geometry.
  - **ST\_GeometryType** - Devuelve el tipo de geometría del valor de ST\_Geometry.
  - **ST\_HasArc** - Tests if a geometry contains a circular arc
  - **ST\_InteriorRingN** - Devuelva el número de anillos interiores de una geometría poligonal.
  - **ST\_InterpolatePoint** - Devuelve el valor de la dimensión medida de una geometría en el punto cerrado al punto proporcionado.
  - **ST\_Intersection** - Computes a geometry representing the shared portion of geometries A and B.
  - **ST\_IsClosed** - Devuelve TRUE si los puntos de inicio y final de una LINESTRING son coincidentes. Para superficies polédricas si son cerradas (volumétricas).
  - **ST\_IsCollection** - Devuelve True si la Geometría es una colección vacía, polígono vacío, punto vacío etc.
  - **ST\_IsPlanar** - Check if a surface is or not planar
  - **ST\_IsPolygonCCW** - Devuelve true si todos los aros exteriores están orientados hacia la izquierda y todos los aros interiores están orientados hacia la derecha.
  - **ST\_IsPolygonCW** - Devuelve true si todos los aros exteriores están orientados hacia la derecha y todos los aros interiores están orientados en sentido contrario a las agujas del reloj.
  - **ST\_IsSimple** - Devuelve (TRUE) si la geometría no tiene puntos geométricos anómalos, como auto intersecciones o tangencias.
  - **ST\_IsSolid** - Test if the geometry is a solid. No validity check is performed.
  - **ST\_IsValidTrajectory** - Tests if the geometry is a valid trajectory.
  - **ST\_Length\_Spheroid** - Returns the 2D or 3D length/perimeter of a lon/lat geometry on a spheroid.
  - **ST\_LineFromMultiPoint** - Crea una LineString desde una geometría MultiPoint.
  - **ST\_LineInterpolatePoint** - Returns a point interpolated along a line at a fractional location.
  - **ST\_LineInterpolatePoints** - Returns points interpolated along a line at a fractional interval.
  - **ST\_LineSubstring** - Returns the part of a line between two fractional locations.
  - **ST\_LineToCurve** - Converts a linear geometry to a curved geometry.
  - **ST\_LocateBetweenElevations** - Returns the portions of a geometry that lie in an elevation (Z) range.
  - **ST\_M** - Returns the M coordinate of a Point.
  - **ST\_MakeLine** - Crea una cadena de línea desde geometrías de punto, multipunto o de línea.
-

- **ST\_MakePoint** - Creates a 2D, 3DZ or 4D Point.
  - **ST\_MakePolygon** - Creates a Polygon from a shell and optional list of holes.
  - **ST\_MakeSolid** - Cast the geometry into a solid. No check is performed. To obtain a valid solid, the input geometry must be a closed Polyhedral Surface or a closed TIN.
  - **ST\_MakeValid** - Attempts to make an invalid geometry valid without losing vertices.
  - **ST\_MemSize** - Devuelve el tipo de geometría del valor de ST\_Geometry.
  - **ST\_MemUnion** - Aggregate function which unions geometries in a memory-efficient but slower way
  - **ST\_NDims** - Devuelve la dimensión de las coordenadas del valor de ST\_Geometry.
  - **ST\_NPoints** - Devuelve el numero de puntos (vértices) en la geometría.
  - **ST\_NRings** - Devuelva el número de anillos interiores de una geometría poligonal.
  - **ST\_Node** - Nodes a collection of lines.
  - **ST\_NumGeometries** - Devuelve el numero de puntos en la geometría. Funciona con todas las geometrías.
  - **ST\_NumPatches** - Devuelve el número de caras en una superficie poliédrica. Devolverá nulo para geometrías no poliédricas.
  - **ST\_Orientation** - Determine surface orientation
  - **ST\_PatchN** - Devuelve el tipo de geometría del valor de ST\_Geometry.
  - **ST\_PointFromWKB** - Crea una geometría desde un WKB con el SRID dado.
  - **ST\_PointN** - Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString.
  - **ST\_PointOnSurface** - Computes a point guaranteed to lie in a polygon, or on a geometry.
  - **ST\_Points** - Devuelve un MultiPoint que contiene todas las coordenadas de una geometría.
  - **ST\_Polygon** - Creates a Polygon from a LineString with a specified SRID.
  - **ST\_RemovePoint** - Remove a point from a linestring.
  - **ST\_RemoveRepeatedPoints** - Returns a version of a geometry with duplicate points removed.
  - **ST\_Reverse** - Devuelve la geometría con el orden de vértice invertido.
  - **ST\_Rotate** - Rotates a geometry about an origin point.
  - **ST\_RotateX** - Rotates a geometry about the X axis.
  - **ST\_RotateY** - Rotates a geometry about the Y axis.
  - **ST\_RotateZ** - Rotates a geometry about the Z axis.
  - **ST\_Scale** - Scales a geometry by given factors.
  - **ST\_Scroll** - Change start point of a closed LineString.
  - **ST\_SetPoint** - Reemplace el punto de una cadena de línea con un punto dado.
  - **ST\_ShiftLongitude** - Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
  - **ST\_SnapToGrid** - Ajusta todos los puntos de la geometría de entrada a una cuadrícula regular.
  - **ST\_StartPoint** - Returns the first point of a LineString.
  - **ST\_StraightSkeleton** - Compute a straight skeleton from a geometry
  - **ST\_SwapOrdinates** - Returns a version of the given geometry with given ordinate values swapped.
-

- **ST\_SymDifference** - Computes a geometry representing the portions of geometries A and B that do not intersect.
  - **ST\_Tessellate** - Perform surface Tessellation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS
  - **ST\_TransScale** - Translates and scales a geometry by given offsets and factors.
  - **ST\_Translate** - Translates a geometry by given offsets.
  - **ST\_UnaryUnion** - Computes the union of the components of a single geometry.
  - **ST\_Union** - Computes a geometry representing the point-set union of the input geometries.
  - **ST\_Volume** - Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0.
  - **ST\_WrapX** - Wrap a geometry around an X value.
  - **ST\_X** - Returns the X coordinate of a Point.
  - **ST\_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
  - **ST\_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
  - **ST\_Y** - Returns the Y coordinate of a Point.
  - **ST\_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
  - **ST\_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
  - **ST\_Z** - Returns the Z coordinate of a Point.
  - **ST\_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
  - **ST\_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
  - **ST\_Zmflag** - Devuelve la dimensión de las coordenadas del valor de ST\_Geometry.
  - **TG\_Equals** - Devuelve true si dos topogeometries están compuestas de las mismas primitivas topologicas.
  - **TG\_Intersects** - Devuelve verdadero si cualquier par de primitivas de las dos topogeometries se intersectan.
  - **UpdateGeometrySRID** - Updates the SRID of all features in a geometry column, and the table metadata.
  - **geometry\_overlaps\_nd** - Returns TRUE if A's n-D bounding box intersects B's n-D bounding box.
  - **overlaps\_nd\_geometry\_gidx** - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
  - **overlaps\_nd\_gidx\_geometry** - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
  - **overlaps\_nd\_gidx\_gidx** - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.
  - **postgis\_sfcgal\_full\_version** - Returns the full version of SFCGAL in use including CGAL and Boost versions
  - **postgis\_sfcgal\_version** - Returns the version of SFCGAL in use
-

## 12.9 PostGIS Curved Geometry Support Functions

The functions given below are PostGIS functions that can use CIRCULARSTRING, CURVEPOLYGON, and other curved geometry types

- **AddGeometryColumn** - Suprime una columna de geometrías de una tabla espacial.
  - **Box2D** - Returns a BOX2D representing the 2D extent of a geometry.
  - **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
  - **DropGeometryColumn** - Suprime una columna de geometrías de una tabla espacial.
  - **GeometryType** - Devuelve el tipo de geometría del valor de ST\_Geometry.
  - **PostGIS\_AddBBBox** - Add bounding box to the geometry.
  - **PostGIS\_DropBBBox** - Drop the bounding box cache from the geometry.
  - **PostGIS\_HasBBBox** - Returns TRUE if the bbox of this geometry is cached, FALSE otherwise.
  - **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
  - **ST\_Affine** - Apply a 3D affine transformation to a geometry.
  - **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
  - **ST\_AsEWKB** - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
  - **ST\_AsEWKT** - Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.
  - **ST\_AsHEXEWKB** - Returns a Geometry in HEXEWKB format (as text) using either little-endian (NDR) or big-endian (XDR) encoding.
  - **ST\_AsSVG** - Returns SVG path data for a geometry.
  - **ST\_AsText** - Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.
  - **ST\_ClusterDBSCAN** - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
  - **ST\_ClusterWithin** - Aggregate function that clusters input geometries by separation distance.
  - **ST\_ClusterWithinWin** - Window function that returns a cluster id for each input geometry, clustering using separation distance.
  - **ST\_GeomCollFromText** - Creates a GeometryCollection or Multi\* geometry from a set of geometries.
  - **ST\_CoordDim** - Devuelve la dimensión de las coordenadas del valor de ST\_Geometry.
  - **ST\_CurveToLine** - Converts a geometry containing curves to a linear geometry.
  - **ST\_Distance** - Returns the distance between two geometry or geography values.
  - **ST\_Dump** - Returns a set of geometry\_dump rows for the components of a geometry.
  - **ST\_NumPoints** - Devuelve un resumen de texto del contenido de la geometría.
  - **ST\_EndPoint** - Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString.
  - **ST\_EstimatedExtent** - Returns the estimated extent of a spatial table.
  - **ST\_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
  - **ST\_Force2D** - Forzar las geometrías en un "modo de 2 dimensiones".
  - **ST\_ForceCurve** - Relanzar una geometría en su tipo curvo, si corresponde.
-



- **ST\_ForceSFS** - Fuerza las geometrías para usar sólo los tipos de geometría SFS 1.1.
  - **ST\_Force3D** - Forzar las geometrías en modo XYZ. Este es un alias para ST\_Force3DZ.
  - **ST\_Force3DM** - Fuerza las geometrías en modo XYM.
  - **ST\_Force3DZ** - Fuerza las geometrías en modo XYZ.
  - **ST\_Force4D** - Fuerza las geometrías en modo XYZM.
  - **ST\_ForceCollection** - Convertir la geometría en una GEOMETRYCOLLECTION.
  - **ST\_GeoHash** - Return a GeoHash representation of the geometry.
  - **ST\_GeogFromWKB** - Crea una instancia "geography" desde la representación de una geometría en "Well-Known Binary" (WKB) o "Extended Well-Known Binary" (EWKB).
  - **ST\_GeomFromEWKB** - Devuelve un valor específico de ST\_Geometry desde una representación "Extended Well-Known Binary" (EWKB).
  - **ST\_GeomFromEWKT** - Devuelve un valor especificado ST\_Geometry desde una representación "Extended Well-Known Text" (EWKT).
  - **ST\_GeomFromText** - Devuelve un valor específico de ST\_Geometry desde una representación "Extended Well-Known Binary" (EWKB).
  - **ST\_GeomFromWKB** - Crea una instancia de geometría desde la representación de una geometría en "Well-Known Binary" (WKB) y un SRID opcional.
  - **ST\_GeometryN** - Devuelve el tipo de geometría del valor de ST\_Geometry.
  - **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
  - **&<|** - Returns TRUE if A's bounding box overlaps or is below B's.
  - **ST\_HasArc** - Tests if a geometry contains a circular arc
  - **ST\_Intersects** - Tests if two geometries intersect (they have at least one point in common)
  - **ST\_IsClosed** - Devuelve TRUE si los puntos de inicio y final de una LINESTRING son coincidentes. Para superficies poliedricas si son cerradas (volumetricas).
  - **ST\_IsCollection** - Devuelve True si la Geometría es una colección vacía, polígono vacío, punto vacío etc.
  - **ST\_IsEmpty** - Tests if a geometry is empty.
  - **ST\_LineToCurve** - Converts a linear geometry to a curved geometry.
  - **ST\_MemSize** - Devuelve el tipo de geometría del valor de ST\_Geometry.
  - **ST\_NPoints** - Devuelve el numero de puntos (vértices) en la geometría.
  - **ST\_NRings** - Devuelva el número de anillos interiores de una geometría poligonal.
  - **ST\_PointFromWKB** - Crea una geometría desde un WKB con el SRID dado.
  - **ST\_PointN** - Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString.
  - **ST\_Points** - Devuelve un MultiPoint que contiene todas las coordenadas de una geometría.
  - **ST\_Rotate** - Rotates a geometry about an origin point.
  - **ST\_RotateZ** - Rotates a geometry about the Z axis.
  - **ST\_SRID** - Returns the spatial reference identifier for a geometry.
-



- **ST\_Scale** - Scales a geometry by given factors.
- **ST\_SetSRID** - Set the SRID on a geometry.
- **ST\_StartPoint** - Returns the first point of a LineString.
- **ST\_Summary** - Devuelve un resumen de texto del contenido de la geometría.
- **ST\_SwapOrdinates** - Returns a version of the given geometry with given ordinate values swapped.
- **ST\_TransScale** - Translates and scales a geometry by given offsets and factors.
- **ST\_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST\_Translate** - Translates a geometry by given offsets.
- **ST\_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- **ST\_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST\_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- **ST\_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- **ST\_Zmflag** - Devuelve la dimensión de las coordenadas del valor de ST\_Geometry.
- **UpdateGeometrySRID** - Updates the SRID of all features in a geometry column, and the table metadata.
- **~(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).
- **&&** - Returns TRUE if A's 2D bounding box intersects B's 2D bounding box.
- **&&&** - Returns TRUE if A's n-D bounding box intersects B's n-D bounding box.
- **@(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- **@(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- **@(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- **&&(box2df,box2df)** - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- **&&(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- **&&(geometry,box2df)** - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- **&&&(geometry,gidx)** - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- **&&&(gidx,geometry)** - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- **&&&(gidx,gidx)** - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

## 12.10 PostGIS Polyhedral Surface Support Functions

The functions given below are PostGIS functions that can use POLYHEDRALSURFACE, POLYHEDRALSURFACEM geometries

- **AddGeometryColumn** - Suprime una columna de geometrías de una tabla espacial.
  - **Box2D** - Returns a BOX2D representing the 2D extent of a geometry.
  - **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
  - **DropGeometryColumn** - Suprime una columna de geometrías de una tabla espacial.
  - **GeometryType** - Devuelve el tipo de geometría del valor de ST\_Geometry.
  - **PostGIS\_AddBBBox** - Add bounding box to the geometry.
  - **PostGIS\_DropBBBox** - Drop the bounding box cache from the geometry.
  - **PostGIS\_HasBBBox** - Returns TRUE if the bbox of this geometry is cached, FALSE otherwise.
  - **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
  - **ST\_Affine** - Apply a 3D affine transformation to a geometry.
  - **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
  - **ST\_AsEWKB** - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
  - **ST\_AsEWKT** - Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.
  - **ST\_AsHEXEWKB** - Returns a Geometry in HEXEWKB format (as text) using either little-endian (NDR) or big-endian (XDR) encoding.
  - **ST\_AsSVG** - Returns SVG path data for a geometry.
  - **ST\_AsText** - Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.
  - **ST\_ClusterDBSCAN** - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
  - **ST\_ClusterWithin** - Aggregate function that clusters input geometries by separation distance.
  - **ST\_ClusterWithinWin** - Window function that returns a cluster id for each input geometry, clustering using separation distance.
  - **ST\_GeomCollFromText** - Creates a GeometryCollection or Multi\* geometry from a set of geometries.
  - **ST\_CoordDim** - Devuelve la dimensión de las coordenadas del valor de ST\_Geometry.
  - **ST\_CurveToLine** - Converts a geometry containing curves to a linear geometry.
  - **ST\_Distance** - Returns the distance between two geometry or geography values.
  - **ST\_Dump** - Returns a set of geometry\_dump rows for the components of a geometry.
  - **ST\_NumPoints** - Devuelve un resumen de texto del contenido de la geometría.
  - **ST\_EndPoint** - Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString.
  - **ST\_EstimatedExtent** - Returns the estimated extent of a spatial table.
  - **ST\_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
  - **ST\_Force2D** - Forzar las geometrías en un "modo de 2 dimensiones".
  - **ST\_ForceCurve** - Relanzar una geometría en su tipo curvo, si corresponde.
-

- **ST\_ForceSFS** - Fuerza las geometrías para usar sólo los tipos de geometría SFS 1.1.
  - **ST\_Force3D** - Forzar las geometrías en modo XYZ. Este es un alias para ST\_Force3DZ.
  - **ST\_Force3DM** - Fuerza las geometrías en modo XYM.
  - **ST\_Force3DZ** - Fuerza las geometrías en modo XYZ.
  - **ST\_Force4D** - Fuerza las geometrías en modo XYZM.
  - **ST\_ForceCollection** - Convertir la geometría en una GEOMETRYCOLLECTION.
  - **ST\_GeoHash** - Return a GeoHash representation of the geometry.
  - **ST\_GeogFromWKB** - Crea una instancia "geography" desde la representación de una geometría en "Well-Known Binary" (WKB) o "Extended Well-Known Binary" (EWKB).
  - **ST\_GeomFromEWKB** - Devuelve un valor específico de ST\_Geometry desde una representación "Extended Well-Known Binary" (EWKB).
  - **ST\_GeomFromEWKT** - Devuelve un valor especificado ST\_Geometry desde una representación "Extended Well-Known Text" (EWKT).
  - **ST\_GeomFromText** - Devuelve un valor específico de ST\_Geometry desde una representación "Extended Well-Known Binary" (EWKB).
  - **ST\_GeomFromWKB** - Crea una instancia de geometría desde la representación de una geometría en "Well-Known Binary" (WKB) y un SRID opcional.
  - **ST\_GeometryN** - Devuelve el tipo de geometría del valor de ST\_Geometry.
  - **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
  - **&<|** - Returns TRUE if A's bounding box overlaps or is below B's.
  - **ST\_HasArc** - Tests if a geometry contains a circular arc
  - **ST\_Intersects** - Tests if two geometries intersect (they have at least one point in common)
  - **ST\_IsClosed** - Devuelve TRUE si los puntos de inicio y final de una LINESTRING son coincidentes. Para superficies polédricas si son cerradas (volumétricas).
  - **ST\_IsCollection** - Devuelve True si la Geometría es una colección vacía, polígono vacío, punto vacío etc.
  - **ST\_IsEmpty** - Tests if a geometry is empty.
  - **ST\_LineToCurve** - Converts a linear geometry to a curved geometry.
  - **ST\_MemSize** - Devuelve el tipo de geometría del valor de ST\_Geometry.
  - **ST\_NPoints** - Devuelve el número de puntos (vértices) en la geometría.
  - **ST\_NRings** - Devuelva el número de anillos interiores de una geometría poligonal.
  - **ST\_PointFromWKB** - Crea una geometría desde un WKB con el SRID dado.
  - **ST\_PointN** - Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString.
  - **ST\_Points** - Devuelve un MultiPoint que contiene todas las coordenadas de una geometría.
  - **ST\_Rotate** - Rotates a geometry about an origin point.
  - **ST\_RotateZ** - Rotates a geometry about the Z axis.
  - **ST\_SRID** - Returns the spatial reference identifier for a geometry.
-

- **ST\_Scale** - Scales a geometry by given factors.
- **ST\_SetSRID** - Set the SRID on a geometry.
- **ST\_StartPoint** - Returns the first point of a LineString.
- **ST\_Summary** - Devuelve un resumen de texto del contenido de la geometría.
- **ST\_SwapOrdinates** - Returns a version of the given geometry with given ordinate values swapped.
- **ST\_TransScale** - Translates and scales a geometry by given offsets and factors.
- **ST\_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST\_Translate** - Translates a geometry by given offsets.
- **ST\_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- **ST\_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST\_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- **ST\_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- **ST\_Zmflag** - Devuelve la dimensión de las coordenadas del valor de ST\_Geometry.
- **UpdateGeometrySRID** - Updates the SRID of all features in a geometry column, and the table metadata.
- **~(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).
- **&&** - Returns TRUE if A's 2D bounding box intersects B's 2D bounding box.
- **&&&** - Returns TRUE if A's n-D bounding box intersects B's n-D bounding box.
- **@(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- **@(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- **@(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- **&&(box2df,box2df)** - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- **&&(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- **&&(geometry,box2df)** - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- **&&&(geometry,gidx)** - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- **&&&(gidx,geometry)** - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- **&&&(gidx,gidx)** - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

## 12.11 PostGIS Function Support Matrix

Below is an alphabetical listing of spatial specific functions in PostGIS and the kinds of spatial types they work with or OGC/SQL compliance they try to conform to.

- A  means the function works with the type or subtype natively.
- A  means it works but with a transform cast built-in using cast to geometry, transform to a "best srid" spatial ref and then cast back. Results may not be as expected for large areas or areas at poles and may accumulate floating point junk.
- A  means the function works with the type because of a auto-cast to another such as to box3d rather than direct type support.
- A  means the function only available if PostGIS compiled with SFCGAL support.
- A  means the function support is provided by SFCGAL if PostGIS compiled with SFCGAL support, otherwise GEOS/built-in support.
- geom - Basic 2D geometry support (x,y).
- geog - Basic 2D geography support (x,y).
- 2.5D - basic 2D geometries in 3 D/4D space (has Z or M coord).
- PS - Polyhedral surfaces
- T - Triangles and Triangulated Irregular Network surfaces (TIN)

| Function          | geom                                                                                | geog | 2.5D                                                                                | Curves | SQL MM                                                                                | PS                                                                                    | T                                                                                     |
|-------------------|-------------------------------------------------------------------------------------|------|-------------------------------------------------------------------------------------|--------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| Box2D             | ✓                                                                                   |      |                                                                                     | ✓      |                                                                                       | ✓                                                                                     | ✓                                                                                     |
| Box3D             | ✓                                                                                   |      | ✓                                                                                   | ✓      |                                                                                       | ✓                                                                                     | ✓                                                                                     |
| GeometryType      | ✓                                                                                   |      | ✓                                                                                   | ✓      |                                                                                       | ✓                                                                                     | ✓                                                                                     |
| PostGIS_AddBBox   | ✓                                                                                   |      |                                                                                     | ✓      |                                                                                       |                                                                                       |                                                                                       |
| PostGIS_DropBBox  | ✓                                                                                   |      |                                                                                     | ✓      |                                                                                       |                                                                                       |                                                                                       |
| PostGIS_HasBBox   | ✓                                                                                   |      |                                                                                     | ✓      |                                                                                       |                                                                                       |                                                                                       |
| ST_3DArea         |  |      |  |        |  |  |  |
| ST_3DClosestPoint | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                       | ✓                                                                                     |                                                                                       |
| ST_3DConvexHull   |  |      |  |        |                                                                                       |  |  |
| ST_3DDFullyWithi  | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                       | ✓                                                                                     |                                                                                       |
| ST_3DDWithin      | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                     | ✓                                                                                     |                                                                                       |
| ST_3DDifference   |  |      |  |        |  |  |  |
| ST_3DDistance     | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                     | ✓                                                                                     |                                                                                       |
| ST_3DExtent       | ✓                                                                                   |      | ✓                                                                                   | ✓      |                                                                                       | ✓                                                                                     | ✓                                                                                     |

| Function                | geom                                                                                | geog | 2.5D                                                                                | Curves | SQL MM                                                                              | PS                                                                                    | T                                                                                     |
|-------------------------|-------------------------------------------------------------------------------------|------|-------------------------------------------------------------------------------------|--------|-------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| ST_3DIntersection       |    |      |    |        |  |    |    |
| ST_3DIntersects         | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                   | ✓                                                                                     | ✓                                                                                     |
| ST_3DLength             | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                   |                                                                                       |                                                                                       |
| ST_LineInterpolatePoint | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_3DLongestLine        | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     | ✓                                                                                     |                                                                                       |
| ST_3DMakeBox            | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_3DMaxDistance        | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     | ✓                                                                                     |                                                                                       |
| ST_3DPerimeter          | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                   |                                                                                       |                                                                                       |
| ST_3DShortestLine       | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     | ✓                                                                                     |                                                                                       |
| ST_3DUnion              |    |      |    |        |  |    |    |
| ST_AddMeasure           | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_AddPoint             | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_Affine               | ✓                                                                                   |      | ✓                                                                                   | ✓      |                                                                                     | ✓                                                                                     | ✓                                                                                     |
| ST_AlphaShape           |   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_Angle                | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_ApproximateMLine     |  | Axis |  |        |                                                                                     |  |  |
| ST_Area                 | ✓                                                                                   | ✓    |                                                                                     |        | ✓                                                                                   | ✓                                                                                     |                                                                                       |
| ST_AsBinary             | ✓                                                                                   | ✓    | ✓                                                                                   | ✓      | ✓                                                                                   | ✓                                                                                     | ✓                                                                                     |
| ST_AsEWKB               | ✓                                                                                   |      | ✓                                                                                   | ✓      |                                                                                     | ✓                                                                                     | ✓                                                                                     |
| ST_AsEWKT               | ✓                                                                                   | ✓    | ✓                                                                                   | ✓      |                                                                                     | ✓                                                                                     | ✓                                                                                     |
| ST_AsEncodedPolyline    | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_AsFlatGeobuf         |  |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_AsGML                | ✓                                                                                   | ✓    | ✓                                                                                   |        | ✓                                                                                   | ✓                                                                                     | ✓                                                                                     |
| ST_AsGeoJSON            | ✓                                                                                   | ✓    | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_AsGeobuf             |  |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_AsHEXEWKB            | ✓                                                                                   |      | ✓                                                                                   | ✓      |                                                                                     |                                                                                       |                                                                                       |
| ST_AsKML                | ✓                                                                                   | ✓    | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_AsLatLonText         | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_AsMARC21             | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_AsMVT                |  |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_AsMVTGeom            | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_AsSVG                | ✓                                                                                   | ✓    |                                                                                     | ✓      |                                                                                     |                                                                                       |                                                                                       |

| Function                        | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|---------------------------------|------|------|------|--------|--------|----|---|
| ST_AsTWKB                       | ✓    |      |      |        |        |    |   |
| ST_AsText                       | ✓    | ✓    |      | ✓      | ✓      |    |   |
| ST_AsX3D                        | ✓    |      | ✓    |        |        | ✓  | ✓ |
| ST_Azimuth                      | ✓    | ✓    |      |        |        |    |   |
| ST_BdMPolyFromText              | ✓    |      |      |        |        |    |   |
| ST_BdPolyFromText               | ✓    |      |      |        |        |    |   |
| ST_Boundary                     | ✓    |      | ✓    |        | ✓      |    |   |
| ST_BoundingDiagonal             | ✓    |      | ✓    |        |        |    |   |
| ST_Box2dFromGeoPolygon          | ✗    |      |      |        |        |    |   |
| ST_Buffer                       | ✓    | 😬    |      |        | ✓      |    |   |
| ST_BuildArea                    | ✓    |      |      |        |        |    |   |
| ST_CPAWithin                    | ✓    |      | ✓    |        |        |    |   |
| ST_Centroid                     | ✓    | ✓    |      |        | ✓      |    |   |
| ST_ChaikinSmooth                | ✓    |      | ✓    |        |        |    |   |
| ST_ClipByBox2D                  | ✓    |      |      |        |        |    |   |
| ST_ClosestPoint                 | ✓    | ✓    |      |        |        |    |   |
| ST_ClosestPointOfApproach       | ✓    |      | ✓    |        |        |    |   |
| ST_ClusterDBSCAN                | ✓    |      |      | ✓      |        |    |   |
| ST_ClusterIntersecting          | ✓    |      |      |        |        |    |   |
| ST_ClusterIntersectingWin       | ✓    |      |      |        |        |    |   |
| ST_ClusterKMeans                | ✓    |      |      |        |        |    |   |
| ST_ClusterWithin                | ✓    |      |      | ✓      |        |    |   |
| ST_ClusterWithinWindow          | ✓    |      |      | ✓      |        |    |   |
| ST_GeomCollFromText             | ✓    |      | ✓    | ✓      |        |    |   |
| ST_CollectionExtract            | ✓    |      |      |        |        |    |   |
| ST_CollectionHomogenize         | ✓    |      |      |        |        |    |   |
| ST_ConcaveHull                  | ✓    |      |      |        |        |    |   |
| ST_ConstrainedDelaunayTriangles | 🔲    |      | 🔲    |        |        |    |   |
| ST_Contains                     | ✓    |      |      |        | ✓      |    |   |
| ST_ContainsProperly             | ✓    |      |      |        |        |    |   |
| ST_ConvexHull                   | ✓    |      | ✓    |        | ✓      |    |   |
| ST_CoordDim                     | ✓    |      | ✓    | ✓      | ✓      | ✓  | ✓ |
| ST_CoverageInvalidates          | ✓    |      |      |        |        |    |   |

| Function                  | geom                                                                                | geog | 2.5D                                                                                | Curves | SQL MM | PS                                                                                    | T                                                                                     |
|---------------------------|-------------------------------------------------------------------------------------|------|-------------------------------------------------------------------------------------|--------|--------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| ST_CoverageSimplification | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_CoverageUnion          | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_CoveredBy              | ✓                                                                                   | ✓    |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_Covers                 | ✓                                                                                   | ✓    |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_Crosses                | ✓                                                                                   |      |                                                                                     |        | ✓      |                                                                                       |                                                                                       |
| ST_CurveToLine            | ✓                                                                                   |      | ✓                                                                                   | ✓      | ✓      |                                                                                       |                                                                                       |
| ST_DFullyWithin           | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_DWithin                | ✓                                                                                   | ✓    |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_DelaunayTriangulation  | ✓                                                                                   |      | ✓                                                                                   |        |        |                                                                                       | ✓                                                                                     |
| ST_Difference             | ✓                                                                                   |      | ✓                                                                                   |        | ✓      |                                                                                       |                                                                                       |
| ST_Dimension              | ✓                                                                                   |      |                                                                                     |        | ✓      | ✓                                                                                     | ✓                                                                                     |
| ST_Disjoint               | ✓                                                                                   |      |                                                                                     |        | ✓      |                                                                                       |                                                                                       |
| ST_Distance               | ✓                                                                                   | ✓    |                                                                                     | ✓      | ✓      |                                                                                       |                                                                                       |
| ST_DistanceCPA            | ✓                                                                                   |      | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_DistanceSphere         | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_DistanceSpheroidal     | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_Dump                   | ✓                                                                                   |      | ✓                                                                                   | ✓      |        | ✓                                                                                     | ✓                                                                                     |
| ST_NumPoints              | ✓                                                                                   |      | ✓                                                                                   | ✓      |        | ✓                                                                                     | ✓                                                                                     |
| ST_NRings                 | ✓                                                                                   |      | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_NumPoints              | ✓                                                                                   |      | ✓                                                                                   |        |        |                                                                                       | ✓                                                                                     |
| ST_EndPoint               | ✓                                                                                   |      | ✓                                                                                   | ✓      | ✓      |                                                                                       |                                                                                       |
| ST_Envelope               | ✓                                                                                   |      |                                                                                     |        | ✓      |                                                                                       |                                                                                       |
| ST_Equals                 | ✓                                                                                   |      |                                                                                     |        | ✓      |                                                                                       |                                                                                       |
| ST_EstimatedExtent        | ✗                                                                                   |      |                                                                                     | ✓      |        |                                                                                       |                                                                                       |
| ST_Expand                 | ✓                                                                                   |      |                                                                                     |        |        | ✓                                                                                     | ✓                                                                                     |
| ST_Extent                 | ✓                                                                                   |      |                                                                                     |        |        | ✓                                                                                     | ✓                                                                                     |
| ST_ExteriorRing           | ✓                                                                                   |      | ✓                                                                                   |        | ✓      |                                                                                       |                                                                                       |
| ST_Extrude                |  |      |  |        |        |  |  |
| ST_FilterByM              | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_FlipCoordinates        | ✓                                                                                   |      | ✓                                                                                   | ✓      |        | ✓                                                                                     | ✓                                                                                     |
| ST_Force2D                | ✓                                                                                   |      | ✓                                                                                   | ✓      |        | ✓                                                                                     |                                                                                       |
| ST_ForceCurve             | ✓                                                                                   |      | ✓                                                                                   | ✓      |        |                                                                                       |                                                                                       |
| ST_ForceLHR               |  |      |  |        |        |  |  |



| Function             | geom    | geog | 2.5D | Curves | SQL MM | PS | T |
|----------------------|---------|------|------|--------|--------|----|---|
| ST_ForcePolygonC     | ✓       |      | ✓    |        |        |    |   |
| ST_ForcePolygonC     | ✓       |      | ✓    |        |        |    |   |
| ST_ForceRHR          | ✓       |      | ✓    |        |        | ✓  |   |
| ST_ForceSFS          | ✓       |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_Force3D           | ✓       |      | ✓    | ✓      |        | ✓  |   |
| ST_Force3DM          | ✓       |      |      | ✓      |        |    |   |
| ST_Force3DZ          | ✓       |      | ✓    | ✓      |        | ✓  |   |
| ST_Force4D           | ✓       |      | ✓    | ✓      |        |    |   |
| ST_ForceCollection   | ✓       |      | ✓    | ✓      |        | ✓  |   |
| ST_FrechetDistance   | ✓       |      |      |        |        |    |   |
| ST_FromFlatGeobuf    |         |      |      |        |        |    |   |
| ST_FromFlatGeobuf    | ✓ table |      |      |        |        |    |   |
| ST_GMLToSQL          | ✓       |      |      |        | ✓      |    |   |
| ST_GeneratePoints    | ✓       |      |      |        |        |    |   |
| ST_GeoHash           | ✓       |      |      | ✓      |        |    |   |
| ST_GeogFromText      |         | ✓    |      |        |        |    |   |
| ST_GeogFromWKB       |         | ✓    |      | ✓      |        |    |   |
| ST_GeographyFromText |         | ✓    |      |        |        |    |   |
| ST_GeomCollFrom      | ✓ :     |      |      |        | ✓      |    |   |
| ST_GeomFromEWI       | ✓       |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_GeomFromEWI       | ✓       |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_GeomFromGML       | ✓       |      | ✓    |        |        | ✓  | ✓ |
| ST_GeomFromGeo       | ✓ n     |      |      |        |        |    |   |
| ST_GeomFromGeo       | ✓ N     |      | ✓    |        |        |    |   |
| ST_GeomFromKML       | ✓       |      | ✓    |        |        |    |   |
| ST_GeomFromMAI       | ✓ .1    |      |      |        |        |    |   |
| ST_GeomFromTWI       | ✓       |      |      |        |        |    |   |
| ST_GeomFromText      | ✓       |      |      | ✓      | ✓      |    |   |
| ST_GeomFromWK        | ✓       |      |      | ✓      | ✓      |    |   |
| ST_GeometricMedi     | ✓       |      | ✓    |        |        |    |   |
| ST_GeometryFrom      | ✓       |      |      |        | ✓      |    |   |
| ST_GeometryN         | ✓       |      | ✓    | ✓      | ✓      | ✓  | ✓ |
| ST_GeometryType      | ✓       |      | ✓    |        | ✓      | ✓  |   |
| >>                   | ✓       |      |      |        |        |    |   |

| Function             | geom | geog   | 2.5D | Curves | SQL MM | PS | T |
|----------------------|------|--------|------|--------|--------|----|---|
| <<                   | ✓    |        |      |        |        |    |   |
| ~                    | ✓    |        |      |        |        |    |   |
| @                    | ✓    |        |      |        |        |    |   |
| =                    | ✓    | ✓      |      | ✓      |        | ✓  |   |
| <<                   | ✓    |        |      |        |        |    |   |
| &>                   | ✓    |        |      |        |        |    |   |
| &<                   | ✓    |        |      | ✓      |        | ✓  |   |
| &<                   | ✓    |        |      |        |        |    |   |
| &>                   | ✓    |        |      |        |        |    |   |
| >>                   | ✓    |        |      |        |        |    |   |
| ~=                   | ✓    |        |      |        |        | ✓  |   |
| ST_HasArc            | ✓    |        | ✓    | ✓      |        |    |   |
| ST_HausdorffDistance | ✓    |        |      |        |        |    |   |
| ST_Hexagon           | ✓    |        |      |        |        |    |   |
| ST_HexagonGrid       | ✓    |        |      |        |        |    |   |
| ST_InteriorRingN     | ✓    |        | ✓    |        | ✓      |    |   |
| ST_InterpolatePoint  | ✓    |        | ✓    |        |        |    |   |
| ST_Intersection      | ✓    | 😄      | ✓    |        | ✓      |    |   |
| ST_Intersects        | ✓    | ✓      |      | ✓      | ✓      |    | ✓ |
| ST_InverseTransform  | ✓    | peline |      |        |        |    |   |
| ST_IsClosed          | ✓    |        | ✓    | ✓      | ✓      | ✓  |   |
| ST_IsCollection      | ✓    |        | ✓    | ✓      |        |    |   |
| ST_IsEmpty           | ✓    |        |      | ✓      | ✓      |    |   |
| ST_IsPlanar          | 🔲    |        | 🔲    |        |        | 🔲  | 🔲 |
| ST_IsPolygonCCW      | ✓    |        | ✓    |        |        |    |   |
| ST_IsPolygonCW       | ✓    |        | ✓    |        |        |    |   |
| ST_IsRing            | ✓    |        |      |        | ✓      |    |   |
| ST_IsSimple          | ✓    |        | ✓    |        | ✓      |    |   |
| ST_IsSolid           | 🔲    |        | 🔲    |        |        | 🔲  | 🔲 |
| ST_IsValid           | ✓    |        |      |        | ✓      |    |   |
| ST_IsValidDetail     | ✓    |        |      |        |        |    |   |
| ST_IsValidReason     | ✓    |        |      |        |        |    |   |
| ST_IsValidTrajectory | ✓    |        | ✓    |        |        |    |   |

| Function                   | geom                                                                                | geog | 2.5D                                                                                | Curves | SQL MM | PS                                                                                    | T                                                                                     |
|----------------------------|-------------------------------------------------------------------------------------|------|-------------------------------------------------------------------------------------|--------|--------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| ST_LargestEmptyCircle      | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_Length                  | ✓                                                                                   | ✓    |                                                                                     |        | ✓      |                                                                                       |                                                                                       |
| ST_Length2D                | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_LengthSpheroid          | ✓                                                                                   |      | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_Letters                 | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_LineCrossingDirection   | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_LineExtend              | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_LineFromEncodedPolyline | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_LineFromMultiPoint      | ✓                                                                                   |      | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_LineFromText            | ✓                                                                                   |      |                                                                                     |        | ✓      |                                                                                       |                                                                                       |
| ST_LineFromWKB             | ✓                                                                                   |      |                                                                                     |        | ✓      |                                                                                       |                                                                                       |
| ST_LineInterpolatePoint    | ✓                                                                                   | ✓    | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_LineInterpolatePoints   | ✓                                                                                   | ✓    | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_LineLocatePoint         | ✓                                                                                   | ✓    |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_LineMerge               | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_LineSubstring           | ✓                                                                                   | ✓    | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_LineToCurve             | ✓                                                                                   |      | ✓                                                                                   | ✓      |        |                                                                                       |                                                                                       |
| ST_LinestringFromBinary    | ✓                                                                                   |      |                                                                                     |        | ✓      |                                                                                       |                                                                                       |
| ST_LocateAlong             | ✓                                                                                   |      |                                                                                     |        | ✓      |                                                                                       |                                                                                       |
| ST_LocateBetween           | ✓                                                                                   |      |                                                                                     |        | ✓      |                                                                                       |                                                                                       |
| ST_LocateBetweenInversions | ✓                                                                                   |      | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_LongestLine             | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_M                       | ✓                                                                                   |      | ✓                                                                                   |        | ✓      |                                                                                       |                                                                                       |
| ST_MLineFromText           | ✓                                                                                   |      |                                                                                     |        | ✓      |                                                                                       |                                                                                       |
| ST_MPointFromText          | ✓                                                                                   |      |                                                                                     |        | ✓      |                                                                                       |                                                                                       |
| ST_MPolyFromText           | ✓                                                                                   |      |                                                                                     |        | ✓      |                                                                                       |                                                                                       |
| ST_MakeBox2D               | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_MakeEnvelope            | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_MakeLine                | ✓                                                                                   |      | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_MakePoint               | ✓                                                                                   |      | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_MakePointM              | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_MakePolygon             | ✓                                                                                   |      | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_MakeSolid               |  |      |  |        |        |  |  |

| Function            | geom                                                                                | geog | 2.5D                                                                                | Curves | SQL MM | PS | T |
|---------------------|-------------------------------------------------------------------------------------|------|-------------------------------------------------------------------------------------|--------|--------|----|---|
| ST_MakeValid        | ✓                                                                                   |      | ✓                                                                                   |        |        |    |   |
| ST_MaxDistance      | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_MaximumInscri    | ✓ Circle                                                                            |      |                                                                                     |        |        |    |   |
| ST_MemSize          | ✓                                                                                   |      | ✓                                                                                   | ✓      |        | ✓  | ✓ |
| ST_MemUnion         | ✓                                                                                   |      | ✓                                                                                   |        |        |    |   |
| ST_MinimumBound     | ✓ Circle                                                                            |      |                                                                                     |        |        |    |   |
| ST_MinimumBound     | ✓ Radius                                                                            |      |                                                                                     |        |        |    |   |
| ST_MinimumCleara    | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_MinimumCleara    | ✓ Line                                                                              |      |                                                                                     |        |        |    |   |
| ST_MinkowskiSum     |    |      |                                                                                     |        |        |    |   |
| ST_Multi            | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_NDims            | ✓                                                                                   |      | ✓                                                                                   |        |        |    |   |
| ST_NPoints          | ✓                                                                                   |      | ✓                                                                                   | ✓      |        | ✓  |   |
| ST_NRings           | ✓                                                                                   |      | ✓                                                                                   | ✓      |        |    |   |
| ST_Node             | ✓                                                                                   |      | ✓                                                                                   |        |        |    |   |
| ST_Normalize        | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_NumGeometries    | ✓                                                                                   |      | ✓                                                                                   |        | ✓      | ✓  | ✓ |
| ST_NumInteriorRin   | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_NumInteriorRin   | ✓                                                                                   |      |                                                                                     |        | ✓      |    |   |
| ST_NumPatches       | ✓                                                                                   |      | ✓                                                                                   |        | ✓      | ✓  |   |
| ST_NumPoints        | ✓                                                                                   |      |                                                                                     |        | ✓      |    |   |
| ST_OffsetCurve      | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_OptimalAlphaS    |  |      |                                                                                     |        |        |    |   |
| ST_OrderingEquals   | ✓                                                                                   |      |                                                                                     |        | ✓      |    |   |
| ST_Orientation      |  |      |  |        |        |    |   |
| ST_OrientedEnvelo   | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_Overlaps         | ✓                                                                                   |      |                                                                                     |        | ✓      |    |   |
| ST_PatchN           | ✓                                                                                   |      | ✓                                                                                   |        | ✓      | ✓  |   |
| ST_Perimeter        | ✓                                                                                   | ✓    |                                                                                     |        | ✓      |    |   |
| ST_Perimeter2D      | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_Point            | ✓                                                                                   |      |                                                                                     |        | ✓      |    |   |
| ST_PointFromGeoHash |                                                                                     |      |                                                                                     |        |        |    |   |
| ST_PointFromText    | ✓                                                                                   |      |                                                                                     |        | ✓      |    |   |

| Function           | geom  | geog | 2.5D | Curves | SQL MM | PS | T |
|--------------------|-------|------|------|--------|--------|----|---|
| ST_PointFromWK     | ✓     |      | ✓    | ✓      | ✓      |    |   |
| ST_PointInsideCirc | ✓     |      |      |        |        |    |   |
| ST_Point           | ✓     |      |      |        |        |    |   |
| ST_PointN          | ✓     |      | ✓    | ✓      | ✓      |    |   |
| ST_PointOnSurface  | ✓     |      | ✓    |        | ✓      |    |   |
| ST_Point           | ✓     |      |      |        |        |    |   |
| ST_Point           | ✓     |      |      |        |        |    |   |
| ST_Points          | ✓     |      | ✓    | ✓      |        |    |   |
| ST_Polygon         | ✓     |      | ✓    |        | ✓      |    |   |
| ST_PolygonFromTe   | ✓     |      |      |        | ✓      |    |   |
| ST_Polygonize      | ✓     |      |      |        |        |    |   |
| ST_Project         | ✓     | ✓    |      |        |        |    |   |
| ST_QuantizeCoordi  | s     |      |      |        |        |    |   |
| ST_ReducePrecision | ✓     |      |      |        |        |    |   |
| ST_Relate          | ✓     |      |      |        | ✓      |    |   |
| ST_RelateMatch     |       |      |      |        |        |    |   |
| ST_RemovePoint     | ✓     |      | ✓    |        |        |    |   |
| ST_RemoveRepeated  | oints |      | ✓    |        |        | ✓  |   |
| ST_Reverse         | ✓     |      | ✓    |        |        | ✓  |   |
| ST_Rotate          | ✓     |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_RotateX         | ✓     |      | ✓    |        |        | ✓  | ✓ |
| ST_RotateY         | ✓     |      | ✓    |        |        | ✓  | ✓ |
| ST_RotateZ         | ✓     |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_SRID            | ✓     |      |      | ✓      | ✓      |    |   |
| ST_Scale           | ✓     |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_Scroll          | ✓     |      | ✓    |        |        |    |   |
| ST_Segmentize      | ✓     | ✓    |      |        |        |    |   |
| ST_SetEffectiveAre | ✓     |      |      |        |        |    |   |
| ST_SetPoint        | ✓     |      | ✓    |        |        |    |   |
| ST_SetSRID         | ✓     |      |      | ✓      |        |    |   |
| ST_SharedPaths     | ✓     |      |      |        |        |    |   |
| ST_ShiftLongitude  | ✓     |      | ✓    |        |        | ✓  | ✓ |
| ST_ShortestLine    | ✓     | ✓    |      |        |        |    |   |
| ST_Simplify        | ✓     |      |      |        |        |    |   |

| Function           | geom                                                                                | geog | 2.5D                                                                                | Curves | SQL MM                                                                                | PS                                                                                    | T                                                                                     |
|--------------------|-------------------------------------------------------------------------------------|------|-------------------------------------------------------------------------------------|--------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| ST_SimplifyPolygo  | ✓ ll                                                                                |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| ST_SimplifyPreserv | ✓ pology                                                                            |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| ST_SimplifyVW      | ✓                                                                                   |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| ST_Snap            | ✓                                                                                   |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| ST_SnapToGrid      | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                       |                                                                                       |                                                                                       |
| ST_Split           | ✓                                                                                   |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| ST_Square          | ✓                                                                                   |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| ST_SquareGrid      | ✓                                                                                   |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| ST_StartPoint      | ✓                                                                                   |      | ✓                                                                                   | ✓      | ✓                                                                                     |                                                                                       |                                                                                       |
| ST_StraightSkeleto |    |      |    |        |                                                                                       |    |    |
| ST_Subdivide       | ✓                                                                                   |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| ST_Summary         | ✓                                                                                   | ✓    |                                                                                     | ✓      |                                                                                       | ✓                                                                                     | ✓                                                                                     |
| ST_SwapOrdinates   | ✓                                                                                   |      | ✓                                                                                   | ✓      |                                                                                       | ✓                                                                                     | ✓                                                                                     |
| ST_SymDifference   | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                     |                                                                                       |                                                                                       |
| ST_Tessellate      |   |      |   |        |                                                                                       |   |   |
| ST_MakeEnvelope    | ✓                                                                                   |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| ST_Touches         | ✓                                                                                   |      |                                                                                     |        | ✓                                                                                     |                                                                                       |                                                                                       |
| ST_TransScale      | ✓                                                                                   |      | ✓                                                                                   | ✓      |                                                                                       |                                                                                       |                                                                                       |
| ST_Transform       | ✓                                                                                   |      |                                                                                     | ✓      | ✓                                                                                     | ✓                                                                                     |                                                                                       |
| ST_TransformPipel  | ✓                                                                                   |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| ST_Translate       | ✓                                                                                   |      | ✓                                                                                   | ✓      |                                                                                       |                                                                                       |                                                                                       |
| ST_TriangulatePoly | ✓                                                                                   |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| ST_UnaryUnion      | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                       |                                                                                       |                                                                                       |
| ST_Union           | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                     |                                                                                       |                                                                                       |
| ST_Volume          |  |      |  |        |  |  |  |
| ST_VoronoiLines    | ✓                                                                                   |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| ST_VoronoiPolygor  | ✓                                                                                   |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| ST_WKBToSQL        | ✓                                                                                   |      |                                                                                     |        | ✓                                                                                     |                                                                                       |                                                                                       |
| ST_WKTToSQL        | ✓                                                                                   |      |                                                                                     |        | ✓                                                                                     |                                                                                       |                                                                                       |
| ST_Within          | ✓                                                                                   |      |                                                                                     |        | ✓                                                                                     |                                                                                       |                                                                                       |
| ST_WrapX           | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                       |                                                                                       |                                                                                       |
| ST_X               | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                     |                                                                                       |                                                                                       |
| ST_XMax            |  |      | ✓                                                                                   | ✓      |                                                                                       |                                                                                       |                                                                                       |

| Function                     | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|------------------------------|------|------|------|--------|--------|----|---|
| ST_XMin                      |      |      |      |        |        |    |   |
| ST_Y                         |      |      |      |        |        |    |   |
| ST_YMax                      |      |      |      |        |        |    |   |
| ST_YMin                      |      |      |      |        |        |    |   |
| ST_Z                         |      |      |      |        |        |    |   |
| ST_ZMax                      |      |      |      |        |        |    |   |
| ST_ZMin                      |      |      |      |        |        |    |   |
| ST_Zmflag                    |      |      |      |        |        |    |   |
| ~(box2df,box2df)             |      |      |      |        |        |    |   |
| ~(box2df,geometry)           |      |      |      |        |        |    |   |
| ~(geometry,box2df)           |      |      |      |        |        |    |   |
| <#>                          |      |      |      |        |        |    |   |
| <<#>>                        |      |      |      |        |        |    |   |
| <<->>                        |      |      |      |        |        |    |   |
| =                            |      |      |      |        |        |    |   |
| <->                          |      |      |      |        |        |    |   |
| &&                           |      |      |      |        |        |    |   |
| &&&                          |      |      |      |        |        |    |   |
| @(box2df,box2df)             |      |      |      |        |        |    |   |
| @(box2df,geometry)           |      |      |      |        |        |    |   |
| @(geometry,box2df)           |      |      |      |        |        |    |   |
| &&(box2df,box2df)            |      |      |      |        |        |    |   |
| &&(box2df,geometry)          |      |      |      |        |        |    |   |
| &&(geometry,box2df)          |      |      |      |        |        |    |   |
| &&&(geometry,gid)            |      |      |      |        |        |    |   |
| &&&(gid,geometry)            |      |      |      |        |        |    |   |
| &&&(gid,gid)                 |      |      |      |        |        |    |   |
| postgis.backend              |      |      |      |        |        |    |   |
| postgis.enable_outdb_rasters |      |      |      |        |        |    |   |
| postgis.gdal_datapath        |      |      |      |        |        |    |   |
| postgis.gdal_enabled_drivers |      |      |      |        |        |    |   |
| postgis.gdal_config_options  |      |      |      |        |        |    |   |
| postgis_sfcgal_full_version  |      |      |      |        |        |    |   |
| postgis_sfcgal_version       |      |      |      |        |        |    |   |
| postgis_srs                  |      |      |      |        |        |    |   |
| postgis_srs_all              |      |      |      |        |        |    |   |

| Function                           | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|------------------------------------|------|------|------|--------|--------|----|---|
| <a href="#">postgis_srs_codes</a>  |      |      |      |        |        |    |   |
| <a href="#">postgis_srs_search</a> | ✓    |      |      |        |        |    |   |

## 12.12 New, Enhanced or changed PostGIS Functions

### 12.12.1 PostGIS Functions new or enhanced in 3.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.4

- [PostGIS\\_GEOS\\_Compiled\\_Version](#) - Availability: 3.4.0 Returns the version number of the GEOS library against which PostGIS was built.
- [ST\\_ClusterIntersectingWin](#) - Availability: 3.4.0 Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
- [ST\\_ClusterWithinWin](#) - Availability: 3.4.0 Window function that returns a cluster id for each input geometry, clustering using separation distance.
- [ST\\_CoverageInvalidEdges](#) - Availability: 3.4.0 - requires GEOS >= 3.12.0 Window function that finds locations where polygons fail to form a valid coverage.
- [ST\\_CoverageSimplify](#) - Availability: 3.4.0 - requires GEOS >= 3.12.0 Window function that simplifies the edges of a polygonal coverage.
- [ST\\_CoverageUnion](#) - Availability: 3.4.0 - requires GEOS >= 3.8.0 Computes the union of a set of polygons forming a coverage by removing shared edges.
- [ST\\_InverseTransformPipeline](#) - Availability: 3.4.0 Return a new geometry with coordinates transformed to a different spatial reference system using the inverse of a defined coordinate transformation pipeline.
- [ST\\_LargestEmptyCircle](#) - Availability: 3.4.0. Computes the largest circle not overlapping a geometry.
- [ST\\_LineExtend](#) - Availability: 3.4.0 Returns a line with the last and first segments extended the specified distance(s).
- [ST\\_TransformPipeline](#) - Availability: 3.4.0 Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.
- [postgis\\_srs](#) - Availability: 3.4.0 Return a metadata record for the requested authority and srid.
- [postgis\\_srs\\_all](#) - Availability: 3.4.0 Return metadata records for every spatial reference system in the underlying Proj database.
- [postgis\\_srs\\_codes](#) - Availability: 3.4.0 Return the list of SRS codes associated with the given authority.
- [postgis\\_srs\\_search](#) - Availability: 3.4.0 Return metadata records for projected coordinate systems that have areas of usage that fully contain the bounds parameter.

Functions enhanced in PostGIS 3.4

- [PostGIS\\_Full\\_Version](#) - Enhanced: 3.4.0 now includes extra PROJ configurations NETWORK\_ENABLED, URL\_ENDPOINT and DATABASE\_PATH of proj.db location Reports full PostGIS version and build configuration infos.
- [PostGIS\\_PROJ\\_Version](#) - Enhanced: 3.4.0 now includes NETWORK\_ENABLED, URL\_ENDPOINT and DATABASE\_PATH of proj.db location Returns the version number of the PROJ4 library.
- [ST\\_AsSVG](#) - Enhanced: 3.4.0 to support all curve types Returns SVG path data for a geometry.



- **ST\_ClosestPoint** - Enhanced: 3.4.0 - Support for geography. Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.
- **ST\_Project** - Enhanced: 3.4.0 Allow geometry arguments and two-point form omitting azimuth. Returns a point projected from a start point by a distance and bearing (azimuth).
- **ST\_ShortestLine** - Enhanced: 3.4.0 - support for geography. Returns the 2D shortest line between two geometries

Functions changed in PostGIS 3.4

- **PostGIS\_Extensions\_Upgrade** - Changed: 3.4.0 to add target\_version argument. Packages and upgrades PostGIS extensions (e.g. postgis\_raster, postgis\_topology, postgis\_sfcgal) to given or latest version.

### 12.12.2 PostGIS Functions new or enhanced in 3.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.3

- **ST\_3DConvexHull** - Disponibilidad: 2.1.0 Computes the 3D convex hull of a geometry.
- **ST\_3DUnion** - Availability: 3.3.0 aggregate variant was added Perform 3D union.
- **ST\_AlphaShape** - Availability: 3.3.0 - requires SFCGAL >= 1.4.1. Computes an Alpha-shape enclosing a geometry
- **ST\_AsMARC21** - Availability: 3.3.0 Returns geometry as a MARC21/XML record with a geographic datafield (034).
- **ST\_GeomFromMARC21** - Availability: 3.3.0, requires libxml2 2.6+ Takes MARC21/XML geographic data as input and returns a PostGIS geometry object.
- **ST\_Letters** - Disponibilidad: 2.1.0 Returns the input letters rendered as geometry with a default start position at the origin and default text height of 100.
- **ST\_OptimalAlphaShape** - Availability: 3.3.0 - requires SFCGAL >= 1.4.1. Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.
- **ST\_SimplifyPolygonHull** - Availability: 3.3.0. Computes a simplified topology-preserving outer or inner hull of a polygonal geometry.
- **ST\_TriangulatePolygon** - Availability: 3.3.0. Computes the constrained Delaunay triangulation of polygons
- **postgis\_sfcgal\_full\_version** - Disponibilidad: 2.1.0 Returns the full version of SFCGAL in use including CGAL and Boost versions

Functions enhanced in PostGIS 3.3

- **ST\_ConcaveHull** - Enhanced: 3.3.0, GEOS native implementation enabled for GEOS 3.11+ Computes a possibly concave geometry that contains all input geometry vertices
- **ST\_LineMerge** - Enhanced: 3.3.0 accept a directed parameter. Return the lines formed by sewing together a MultiLineString.

Functions changed in PostGIS 3.3

- **PostGIS\_Extensions\_Upgrade** - Changed: 3.3.0 support for upgrades from any PostGIS version. Does not work on all systems. Packages and upgrades PostGIS extensions (e.g. postgis\_raster, postgis\_topology, postgis\_sfcgal) to given or latest version.

### 12.12.3 PostGIS Functions new or enhanced in 3.2

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.2

- **ST\_AsFlatGeobuf** - Availability: 3.2.0 Return a FlatGeobuf representation of a set of rows.
- **ST\_FromFlatGeobuf** - Availability: 3.2.0 Reads FlatGeobuf data.
- **ST\_FromFlatGeobufToTable** - Availability: 3.2.0 Creates a table based on the structure of FlatGeobuf data.
- **ST\_NumPoints** - Disponibilidad: 2.2.0 Devuelve un resumen de texto del contenido de la geometría.
- **ST\_Scroll** - Availability: 3.2.0 Change start point of a closed LineString.
- **postgis.gdal\_config\_options** - Disponibilidad: 2.2.0 A string configuration to set options used when working with an out-db raster.

Functions enhanced in PostGIS 3.2

- **ST\_ClusterKMeans** - Enhanced: 3.2.0 Support for max\_radius Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST\_MakeValid** - Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS >= 3.10.0. Attempts to make an invalid geometry valid without losing vertices.
- **ST\_Point** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry. Creates a Point with X, Y and SRID values.
- **ST\_Point** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z and SRID values.
- **ST\_Point** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, M and SRID values.
- **ST\_Point** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z, M and SRID values.
- **ST\_RemovePoint** - Enhanced: 3.2.0 Remove a point from a linestring.
- **ST\_RemoveRepeatedPoints** - Enhanced: 3.2.0 Returns a version of a geometry with duplicate points removed.
- **ST\_StartPoint** - Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString. Returns the first point of a LineString.

Functions changed in PostGIS 3.2

- **ST\_Boundary** - Changed: 3.2.0 support for TIN, does not use geos, does not linearize curves Devuelve el cierre del limite combinatorio de esta geometría.

### 12.12.4 PostGIS Functions new or enhanced in 3.1

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.1

- **ST\_Hexagon** - Disponibilidad: 2.1.0 Returns a single hexagon, using the provided edge size and cell coordinate within the hexagon grid space.
  - **ST\_HexagonGrid** - Disponibilidad: 2.1.0 Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.
-

- **ST\_MaximumInscribedCircle** - Availability: 3.1.0. Computes the largest circle contained within a geometry.
- **ST\_ReducePrecision** - Availability: 3.1.0. Returns a valid geometry with points rounded to a grid tolerance.
- **ST\_Square** - Disponibilidad: 2.1.0 Returns a single square, using the provided edge size and cell coordinate within the square grid space.
- **ST\_SquareGrid** - Disponibilidad: 2.1.0 Returns a set of grid squares and cell indices that completely cover the bounds of the geometry argument.

#### Functions enhanced in PostGIS 3.1

- **ST\_AsEWKT** - Enhanced: 3.1.0 support for optional precision parameter. Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.
- **ST\_ClusterKMeans** - Enhanced: 3.1.0 Support for 3D geometries and weights Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST\_Difference** - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the part of geometry A that does not intersect geometry B.
- **ST\_Intersection** - Enhanced: 3.1.0 accept a gridSize parameter Computes a geometry representing the shared portion of geometries A and B.
- **ST\_MakeEnvelope** - Mejorada: 2.0.0 se agregó el parámetro por defecto opcional srid. Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.
- **ST\_MakeValid** - Enhanced: 3.1.0, added removal of Coordinates with NaN values. Attempts to make an invalid geometry valid without losing vertices.
- **ST\_Subdivide** - Enhanced: 3.1.0 accept a gridSize parameter. Computes a rectilinear subdivision of a geometry.
- **ST\_SymDifference** - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the portions of geometries A and B that do not intersect.
- **ST\_UnaryUnion** - Enhanced: 3.1.0 accept a gridSize parameter. Computes the union of the components of a single geometry.
- **ST\_Union** - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the point-set union of the input geometries.

#### Functions changed in PostGIS 3.1

- **ST\_Force3D** - Changed: 3.1.0. Added support for supplying a non-zero Z value. Forzar las geometrías en modo XYZ. Este es un alias para ST\_Force3DZ.
- **ST\_Force3DM** - Changed: 3.1.0. Added support for supplying a non-zero M value. Fuerza las geometrías en modo XYM.
- **ST\_Force3DZ** - Changed: 3.1.0. Added support for supplying a non-zero Z value. Fuerza las geometrías en modo XYZ.
- **ST\_Force4D** - Changed: 3.1.0. Added support for supplying non-zero Z and M values. Fuerza las geometrías en modo XYZM.

### 12.12.5 PostGIS Functions new or enhanced in 3.0

The functions given below are PostGIS functions that were added or enhanced.

#### Functions new in PostGIS 3.0

- **ST\_ConstrainedDelaunayTriangles** - Disponibilidad: 2.1.0 Return a constrained Delaunay triangulation around the given input geometry.
- **ST\_LineInterpolatePoint** - Disponibilidad: 2.0.0 Returns a point interpolated along a 3D line at a fractional location.

- **ST\_MakeEnvelope** - Disponibilidad: 2.1.0 Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.

#### Functions enhanced in PostGIS 3.0

- **ST\_AsMVT** - Enhanced: 3.0 - added support for Feature ID. Aggregate function returning a MVT representation of a set of rows.
- **ST\_Contains** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in A, and their interiors have a point in common
- **ST\_ContainsProperly** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in the interior of A
- **ST\_CoveredBy** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of A lies in B
- **ST\_Covers** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in A
- **ST\_Crosses** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have some, but not all, interior points in common
- **ST\_CurveToLine** - Enhanced: 3.0.0 implemented a minimum number of segments per linearized arc to prevent topological collapse. Converts a geometry containing curves to a linear geometry.
- **ST\_Disjoint** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have no points in common
- **ST\_Equals** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries include the same set of points
- **ST\_GeneratePoints** - Enhanced: 3.0.0, added seed parameter Generates random points contained in a Polygon or MultiPolygon.
- **ST\_GeomFromGeoJSON** - Enhanced: 3.0.0 parsed geometry defaults to SRID=4326 if not specified otherwise. Toma como entrada una representación geojson de una geometría y devuelve un objeto geométrico PostGIS
- **ST\_LocateBetween** - Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. Returns the portions of a geometry that match a measure range.
- **ST\_LocateBetweenElevations** - Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. Returns the portions of a geometry that lie in an elevation (Z) range.
- **ST\_Overlaps** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other
- **ST\_Relate** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix
- **ST\_Segmentize** - Enhanced: 3.0.0 Segmentize geometry now produces equal-length subsegments Returns a modified geometry/geography having no segment longer than a given distance.
- **ST\_Touches** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have at least one point in common, but their interiors do not intersect
- **ST\_Within** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of A lies in B, and their interiors have a point in common

#### Functions changed in PostGIS 3.0

- **PostGIS\_Extensions\_Upgrade** - Changed: 3.0.0 to repackage loose extensions and support postgis\_raster. Packages and upgrades PostGIS extensions (e.g. postgis\_raster, postgis\_topology, postgis\_sfcgal) to given or latest version.
- **ST\_3DDistance** - Changed: 3.0.0 - SFCGAL version removed Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units.

- **ST\_3DIntersects** - Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs. Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- **ST\_Area** - Changed: 3.0.0 - does not depend on SFCGAL anymore. Returns the area of a polygonal geometry.
- **ST\_AsGeoJSON** - Changed: 3.0.0 support records as input Return a geometry as a GeoJSON element.
- **ST\_AsGeoJSON** - Changed: 3.0.0 output SRID if not EPSG:4326. Return a geometry as a GeoJSON element.
- **ST\_AsKML** - Changed: 3.0.0 - Removed the "versioned" variant signature Return the geometry as a KML element.
- **ST\_Distance** - Changed: 3.0.0 - does not depend on SFCGAL anymore. Returns the distance between two geometry or geography values.
- **ST\_Intersection** - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the shared portion of geometries A and B.
- **ST\_Intersects** - Changed: 3.0.0 SFCGAL version removed and native support for 2D TINs added. Tests if two geometries intersect (they have at least one point in common)
- **ST\_Union** - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the point-set union of the input geometries.

### 12.12.6 PostGIS Functions new or enhanced in 2.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.5

- **PostGIS\_Extensions\_Upgrade** - Availability: 2.5.0 Packages and upgrades PostGIS extensions (e.g. postgis\_raster, postgis\_topology, postgis\_sfcgal) to given or latest version.
- **ST\_Angle** - Availability: 2.5.0 Returns the angle between two vectors defined by 3 or 4 points, or 2 lines.
- **ST\_ChaikinSmoothing** - Availability: 2.5.0 Returns a smoothed version of a geometry, using the Chaikin algorithm
- **ST\_FilterByM** - Availability: 2.5.0 Removes vertices based on their M value
- **ST\_LineInterpolatePoints** - Availability: 2.5.0 Returns points interpolated along a line at a fractional interval.
- **ST\_OrientedEnvelope** - Availability: 2.5.0. Returns a minimum-area rectangle containing a geometry.
- **ST\_QuantizeCoordinates** - Availability: 2.5.0 Sets least significant bits of coordinates to zero

Functions enhanced in PostGIS 2.5

- **ST\_AsMVT** - Enhanced: 2.5.0 - added support parallel query. Aggregate function returning a MVT representation of a set of rows.
- **ST\_AsText** - Enhanced: 2.5 - optional parameter precision introduced. Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.
- **ST\_Buffer** - Enhanced: 2.5.0 - ST\_Buffer geometry support was enhanced to allow for side buffering specification side=both|left|right. Computes a geometry covering all points within a given distance from a geometry.
- **ST\_GeomFromGeoJSON** - Enhanced: 2.5.0 can now accept json and jsonb as inputs. Toma como entrada una representación geojson de una geometría y devuelve un objeto geométrico PostGIS
- **ST\_GeometricMedian** - Enhanced: 2.5.0 Added support for M as weight of points. Returns the geometric median of a Multi-Point.
- **ST\_Intersects** - Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION. Tests if two geometries intersect (they have at least one point in common)

- **ST\_OffsetCurve** - Enhanced: 2.5 - added support for GEOMETRYCOLLECTION and MULTILINESTRING Returns an offset line at a given distance and side from an input line.
- **ST\_Scale** - Enhanced: 2.5.0 support for scaling relative to a local origin (origin parameter) was introduced. Scales a geometry by given factors.
- **ST\_Split** - Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced. Returns a collection of geometries created by splitting a geometry by another geometry.
- **ST\_Subdivide** - Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5. Computes a rectilinear subdivision of a geometry.

### 12.12.7 PostGIS Functions new or enhanced in 2.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.4

- **ST\_AsGeobuf** - Availability: 2.4.0 Return a Geobuf representation of a set of rows.
- **ST\_AsMVT** - Availability: 2.4.0 Aggregate function returning a MVT representation of a set of rows.
- **ST\_AsMVTGeom** - Availability: 2.4.0 Transforms a geometry into the coordinate space of a MVT tile.
- **ST\_Centroid** - Availability: 2.4.0 support for geography was introduced. Returns the geometric center of a geometry.
- **ST\_ForcePolygonCCW** - Availability: 2.4.0 Orienta todos los aros exteriores en sentido contrario a las agujas del reloj y todos los aros interiores en sentido horario.
- **ST\_ForcePolygonCW** - Availability: 2.4.0 Orienta todos los anillos exteriores en el sentido de las agujas del reloj y todos los anillos interiores en sentido contrario a las agujas del reloj.
- **ST\_FrechetDistance** - Availability: 2.4.0 - requires GEOS >= 3.7.0 Returns the Fréchet distance between two geometries.
- **ST\_IsPolygonCCW** - Disponibilidad: 2.2.0 Devuelve true si todos los aros exteriores están orientados hacia la izquierda y todos los aros interiores están orientados hacia la derecha.
- **ST\_IsPolygonCW** - Disponibilidad: 2.2.0 Devuelve true si todos los aros exteriores están orientados hacia la derecha y todos los aros interiores están orientados en sentido contrario a las agujas del reloj.

Functions enhanced in PostGIS 2.4

- **ST\_AsTWKB** - Enhanced: 2.4.0 memory and speed improvements. Returns the geometry as TWKB, aka "Tiny Well-Known Binary"
- **ST\_Covers** - Enhanced: 2.4.0 Support for polygon in polygon and line in polygon added for geography type Tests if every point of B lies in A
- **ST\_CurveToLine** - Enhanced: 2.4.0 added support for max-deviation and max-angle tolerance, and for symmetric output. Converts a geometry containing curves to a linear geometry.
- **ST\_Project** - Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth. Returns a point projected from a start point by a distance and bearing (azimuth).
- **ST\_Reverse** - Mejorada: 2.4.0 se introdujo el soporte para curvas. Devuelve la geometría con el orden de vértice invertido.

Functions changed in PostGIS 2.4

- **=** - Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use `ST_Equals`. Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **ST\_Node** - Changed: 2.4.0 this function uses GEOSNode internally instead of GEOSUnaryUnion. This may cause the resulting linestrings to have a different order and direction compared to PostGIS < 2.4. Nodes a collection of lines.



## 12.12.8 PostGIS Functions new or enhanced in 2.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.3

- **&&&(geometry,gidx)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- **&&&(gidx,geometry)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- **&&&(gidx,gidx)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.
- **&&(box2df,box2df)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- **&&(box2df,geometry)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- **&&(geometry,box2df)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- **@(box2df,box2df)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- **@(box2df,geometry)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- **@(geometry,box2df)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- **ST\_ClusterDBSCAN** - Availability: 2.3.0 Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- **ST\_ClusterKMeans** - Availability: 2.3.0 Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST\_GeneratePoints** - Disponibilidad: 2.3.0 Generates random points contained in a Polygon or MultiPolygon.
- **ST\_GeometricMedian** - Disponibilidad: 2.3.0 Returns the geometric median of a MultiPoint.
- **ST\_MakeLine** - Disponibilidad: 2.3.0 - Se introdujo soporte para elementos de entrada multipunto Crea una cadena de línea desde geometrías de punto, multipunto o de línea.
- **ST\_MinimumBoundingRadius** - Disponibilidad: 2.3.0 Returns the center point and radius of the smallest circle that contains a geometry.
- **ST\_MinimumClearance** - Disponibilidad: 2.3.0 Returns the minimum clearance of a geometry, a measure of a geometry's robustness.
- **ST\_MinimumClearanceLine** - Availability: 2.3.0 - requires GEOS >= 3.6.0 Returns the two-point LineString spanning a geometry's minimum clearance.
- **ST\_Normalize** - Disponibilidad: 2.3.0 Devuelve la geometría en su forma canónica.
- **ST\_Points** - Disponibilidad: 2.3.0 Devuelve un MultiPoint que contiene todas las coordenadas de una geometría.
- **ST\_VoronoiLines** - Disponibilidad: 2.3.0 Returns the boundaries of the Voronoi diagram of the vertices of a geometry.
- **ST\_VoronoiPolygons** - Disponibilidad: 2.3.0 Returns the cells of the Voronoi diagram of the vertices of a geometry.
- **ST\_WrapX** - Availability: 2.3.0 requires GEOS Wrap a geometry around an X value.

- **~(box2df,box2df)** - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- **~(geometry,box2df)** - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).

#### Functions enhanced in PostGIS 2.3

- **ST\_Contains** - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of B lies in A, and their interiors have a point in common
- **ST\_Covers** - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of B lies in A
- **ST\_Expand** - Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions. Returns a bounding box expanded from another bounding box or a geometry.
- **ST\_Intersects** - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if two geometries intersect (they have at least one point in common)
- **ST\_Segmentize** - Enhanced: 2.3.0 Segmentize geography now produces equal-length subsegments Returns a modified geometry/geography having no segment longer than a given distance.
- **ST\_Transform** - Enhanced: 2.3.0 support for direct PROJ.4 text was introduced. Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST\_Within** - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of A lies in B, and their interiors have a point in common

#### Functions changed in PostGIS 2.3

- **ST\_PointN** - Cambiado: 2.3.0: indexación negativa disponible (-1 es el último punto) Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString.

## 12.12.9 PostGIS Functions new or enhanced in 2.2

The functions given below are PostGIS functions that were added or enhanced.

#### Functions new in PostGIS 2.2

- **<<#>>** - Availability: 2.2.0 -- KNN only available for PostgreSQL 9.1+ Returns the n-D distance between A and B bounding boxes.
- **<<->>** - Availability: 2.2.0 -- KNN only available for PostgreSQL 9.1+ Returns the n-D distance between the centroids of A and B bounding boxes.
- **ST\_3DDifference** - Disponibilidad: 2.2.0 Perform 3D difference
- **ST\_3DUnion** - Disponibilidad: 2.2.0 Perform 3D union.
- **ST\_ApproximateMedialAxis** - Disponibilidad: 2.2.0 Compute the approximate medial axis of an areal geometry.
- **ST\_AsEncodedPolyline** - Disponibilidad: 2.2.0 Returns an Encoded Polyline from a LineString geometry.
- **ST\_AsTWKB** - Disponibilidad: 2.2.0 Returns the geometry as TWKB, aka "Tiny Well-Known Binary"
- **ST\_BoundingDiagonal** - Disponibilidad: 2.2.0 Devuelve la diagonal del cuadro delimitador de la geometría suministrada.
- **ST\_CPAWithin** - Availability: 2.2.0 Tests if the closest point of approach of two trajectories is within the specified distance.



- **ST\_ClipByBox2D** - Availability: 2.2.0 Computes the portion of a geometry falling within a rectangle.
- **ST\_ClosestPointOfApproach** - Availability: 2.2.0 Returns a measure at the closest point of approach of two trajectories.
- **ST\_ClusterIntersecting** - Availability: 2.2.0 Aggregate function that clusters input geometries into connected sets.
- **ST\_ClusterWithin** - Availability: 2.2.0 Aggregate function that clusters input geometries by separation distance.
- **ST\_DistanceCPA** - Availability: 2.2.0 Returns the distance between the closest point of approach of two trajectories.
- **ST\_ForceCurve** - Disponibilidad: 2.2.0 Relanzar una geometría en su tipo curvo, si corresponde.
- **ST\_IsPlanar** - Availability: 2.2.0: This was documented in 2.1.0 but got accidentally left out in 2.1 release. Check if a surface is or not planar
- **ST\_IsSolid** - Disponibilidad: 2.2.0 Test if the geometry is a solid. No validity check is performed.
- **ST\_IsValidTrajectory** - Availability: 2.2.0 Tests if the geometry is a valid trajectory.
- **ST\_LineFromEncodedPolyline** - Disponibilidad: 2.2.0 Crea un LineString desde una polilínea codificada.
- **ST\_MakeSolid** - Disponibilidad: 2.2.0 Cast the geometry into a solid. No check is performed. To obtain a valid solid, the input geometry must be a closed Polyhedral Surface or a closed TIN.
- **ST\_RemoveRepeatedPoints** - Disponibilidad: 2.2.0 Returns a version of a geometry with duplicate points removed.
- **ST\_SetEffectiveArea** - Disponibilidad: 2.2.0 Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.
- **ST\_SimplifyVW** - Disponibilidad: 2.2.0 Returns a simplified version of a geometry, using the Visvalingam-Whyatt algorithm
- **ST\_Subdivide** - Availability: 2.2.0 Computes a rectilinear subdivision of a geometry.
- **ST\_SwapOrdinates** - Disponibilidad: 2.2.0 Returns a version of the given geometry with given ordinate values swapped.
- **ST\_Volume** - Disponibilidad: 2.2.0 Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0.
- **postgis.enable\_outdb\_rasters** - Disponibilidad: 2.2.0 A boolean configuration option to enable access to out-db raster bands.
- **postgis.gdal\_datapath** - Disponibilidad: 2.2.0 A configuration option to assign the value of GDAL's GDAL\_DATA option. If not set, the environmentally set GDAL\_DATA variable is used.
- **postgis.gdal\_enabled\_drivers** - Disponibilidad: 2.2.0 A configuration option to set the enabled GDAL drivers in the PostGIS environment. Affects the GDAL configuration variable GDAL\_SKIP.
- **||** - Availability: 2.2.0. Index-supported only available for PostgreSQL 9.5+ Returns the distance between A and B trajectories at their closest point of approach.

#### Functions enhanced in PostGIS 2.2

- **<->** - Enhanced: 2.2.0 -- True KNN ("K nearest neighbor") behavior for geometry and geography for PostgreSQL 9.5+. Note for geography KNN is based on sphere rather than spheroid. For PostgreSQL 9.4 and below, geography support is new but only supports centroid box. Returns the 2D distance between A and B.
- **ST\_Area** - Enhanced: 2.2.0 - measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires PROJ >= 4.9.0 to take advantage of the new feature. Returns the area of a polygonal geometry.
- **ST\_AsX3D** - Enhanced: 2.2.0: Support for GeoCoordinates and axis (x/y, long/lat) flipping. Look at options for details. Returns a Geometry in X3D xml node element format: ISO-IEC-19776-1.2-X3DEncodings-XML
- **ST\_Azimuth** - Enhanced: 2.2.0 measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires PROJ >= 4.9.0 to take advantage of the new feature. Returns the north-based azimuth of a line between two points.

- **ST\_Distance** - Enhanced: 2.2.0 - measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires PROJ >= 4.9.0 to take advantage of the new feature. Returns the distance between two geometry or geography values.
- **ST\_Scale** - Enhanced: 2.2.0 support for scaling all dimension (factor parameter) was introduced. Scales a geometry by given factors.
- **ST\_Split** - Enhanced: 2.2.0 support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced. Returns a collection of geometries created by splitting a geometry by another geometry.
- **ST\_Summary** - Mejorado: 2.2.0 agregó soporte para TIN y curvas Devuelve un resumen de texto del contenido de la geometría.

#### Functions changed in PostGIS 2.2

- **<->** - Changed: 2.2.0 -- For PostgreSQL 9.5 users, old Hybrid syntax may be slower, so you'll want to get rid of that hack if you are running your code only on PostGIS 2.2+ 9.5+. See examples below. Returns the 2D distance between A and B.
- **ST\_3DClosestPoint** - Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z. Returns the 3D point on g1 that is closest to g2. This is the first point of the 3D shortest line.
- **ST\_3DDistance** - Changed: 2.2.0 - In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z. Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units.
- **ST\_3DLongestLine** - Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z. Returns the 3D longest line between two geometries
- **ST\_3DMaxDistance** - Changed: 2.2.0 - In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z. Returns the 3D cartesian maximum distance (based on spatial ref) between two geometries in projected units.
- **ST\_3DShortestLine** - Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z. Returns the 3D shortest line between two geometries
- **ST\_DistanceSphere** - Changed: 2.2.0 In prior versions this used to be called ST\_Distance\_Sphere Returns minimum distance in meters between two lon/lat geometries using a spherical earth model.
- **ST\_DistanceSpheroid** - Changed: 2.2.0 In prior versions this was called ST\_Distance\_Spheroid Returns the minimum distance between two lon/lat geometries using a spheroidal earth model.
- **ST\_Equals** - Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal Tests if two geometries include the same set of points
- **ST\_LengthSpheroid** - Changed: 2.2.0 In prior versions this was called ST\_Length\_Spheroid and had the alias ST\_3DLength\_Spheroid Returns the 2D or 3D length/perimeter of a lon/lat geometry on a spheroid.
- **ST\_MemSize** - Changed: 2.2.0 name changed to ST\_MemSize to follow naming convention. Devuelve el tipo de geometría del valor de ST\_Geometry.
- **ST\_PointInsideCircle** - Changed: 2.2.0 In prior versions this was called ST\_Point\_Inside\_Circle Tests if a point geometry is inside a circle defined by a center and radius

#### 12.12.10 PostGIS Functions new or enhanced in 2.1

The functions given below are PostGIS functions that were added or enhanced.

##### Functions new in PostGIS 2.1

- **ST\_3DArea** - Disponibilidad: 2.1.0 Computes area of 3D surface geometries. Will return 0 for solids.

- **ST\_3DIntersection** - Disponibilidad: 2.1.0 Perform 3D intersection
- **ST\_Box2dFromGeoHash** - Disponibilidad: 2.1.0 Devuelve un BOX2D de una cadena de GeoHash.
- **ST\_DelaunayTriangles** - Disponibilidad: 2.1.0 Returns the Delaunay triangulation of the vertices of a geometry.
- **ST\_Extrude** - Disponibilidad: 2.1.0 Extrude a surface to a related volume
- **ST\_ForceLHR** - Disponibilidad: 2.1.0 Force LHR orientation
- **ST\_GeomFromGeoHash** - Disponibilidad: 2.1.0 Devuelve una geometría de una cadena de GeoHash.
- **ST\_MinkowskiSum** - Disponibilidad: 2.1.0 Performs Minkowski sum
- **ST\_Orientation** - Disponibilidad: 2.1.0 Determine surface orientation
- **ST\_PointFromGeoHash** - Disponibilidad: 2.1.0 Devuelve un punto de una cadena de GeoHash.
- **ST\_StraightSkeleton** - Disponibilidad: 2.1.0 Compute a straight skeleton from a geometry
- **ST\_Tessellate** - Disponibilidad: 2.1.0 Perform surface Tessellation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS
- **postgis.backend** - Disponibilidad: 2.1.0 The backend to service a function where GEOS and SFCGAL overlap. Options: geos or sfcgal. Defaults to geos.
- **postgis\_sfcgal\_version** - Disponibilidad: 2.1.0 Returns the version of SFCGAL in use

#### Functions enhanced in PostGIS 2.1

- **ST\_AsGML** - Enhanced: 2.1.0 id support was introduced, for GML 3. Return the geometry as a GML version 2 or 3 element.
- **ST\_Boundary** - Mejorado: 2.1.0 Se ha introducido soporte para Triangle Devuelve el cierre del limite combinatorio de esta geometría.
- **ST\_DWithin** - Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details. Tests if two geometries are within a given distance
- **ST\_DWithin** - Enhanced: 2.1.0 support for curved geometries was introduced. Tests if two geometries are within a given distance
- **ST\_Distance** - Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details. Returns the distance between two geometry or geography values.
- **ST\_Distance** - Enhanced: 2.1.0 - support for curved geometries was introduced. Returns the distance between two geometry or geography values.
- **ST\_MakeValid** - Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT. Attempts to make an invalid geometry valid without losing vertices.
- **ST\_NumPoints** - Enhanced: 2.1.0 Faster speed. Reimplemented as native-C. Devuelve un resumen de texto del contenido de la geometría.
- **ST\_Segmentize** - Mejorada: 2.1.0 se introdujo el soporte para geography. Returns a modified geometry/geography having no segment longer than a given distance.
- **ST\_Summary** - Mejorada: 2.1.0 Indicador S para señalar si tiene un sistema de referencia espacial conocido Devuelve un resumen de texto del contenido de la geometría.

#### Functions changed in PostGIS 2.1

- **ST\_EstimatedExtent** - Changed: 2.1.0. Up to 2.0.x this was called ST\_Estimated\_Extent. Returns the estimated extent of a spatial table.

- **ST\_Force2D** - Cambiado: 2.1.0. Hasta la 2.0.x esto se llamaba ST\_Force\_2D. Forzar las geometrías en un "modo de 2 dimensiones".
- **ST\_Force3D** - Cambiado: 2.1.0. Hasta la 2.0.x esto se llamaba ST\_Force\_3D. Forzar las geometrías en modo XYZ. Este es un alias para ST\_Force3DZ.
- **ST\_Force3DM** - Cambiado: 2.1.0. Hasta la 2.0.x esto se llamaba ST\_Force\_3DM. Fuerza las geometrías en modo XYM.
- **ST\_Force3DZ** - Cambiado: 2.1.0. Hasta la 2.0.x esto se llamaba ST\_Force\_3DZ. Fuerza las geometrías en modo XYZ.
- **ST\_Force4D** - Cambiado: 2.1.0. Hasta la 2.0.x esto se llamaba ST\_Force\_4D. Fuerza las geometrías en modo XYZM.
- **ST\_ForceCollection** - Cambiado: 2.1.0. Hasta la 2.0.x esto se llamaba ST\_Force\_Collection. Convertir la geometría en una GEOMETRYCOLLECTION.
- **ST\_LineInterpolatePoint** - Cambiado: 2.1.0. Hasta 2.0. x esto se llamaba ST\_Line\_Interpolate\_Point. Returns a point interpolated along a line at a fractional location.
- **ST\_LineLocatePoint** - Modificado: 2.1.0. Hasta 2.0.x esto se llamaba ST\_Line\_Locate\_Point. Returns the fractional location of the closest point on a line to a point.
- **ST\_LineSubstring** - Modificado: 2.1.0. Hasta 2.0.x esto se llamaba ST\_Line\_Substring. Returns the part of a line between two fractional locations.
- **ST\_Segmentize** - Changed: 2.1.0 As a result of the introduction of geography support, the usage ST\_Segmentize('LINESTRING(1 2, 3 4)', 0.5) causes an ambiguous function error. The input needs to be properly typed as a geometry or geography. Use ST\_GeomFromText, ST\_GeogFromText or a cast to the required type (e.g. ST\_Segmentize('LINESTRING(1 2, 3 4)::geometry, 0.5) ) Returns a modified geometry/geography having no segment longer than a given distance.

### 12.12.11 PostGIS Functions new or enhanced in 2.0

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.0

- **&&&** - Disponibilidad: 2.0.0 Returns TRUE if A's n-D bounding box intersects B's n-D bounding box.
- **<#>** - Availability: 2.0.0 -- KNN only available for PostgreSQL 9.1+ Returns the 2D distance between A and B bounding boxes.
- **<->** - Availability: 2.0.0 -- Weak KNN provides nearest neighbors based on geometry centroid distances instead of true distances. Exact results for points, inexact for all other types. Available for PostgreSQL 9.1+ Returns the 2D distance between A and B.
- **ST\_3DClosestPoint** - Disponibilidad: 2.0.0 Returns the 3D point on g1 that is closest to g2. This is the first point of the 3D shortest line.
- **ST\_3DDFullyWithin** - Availability: 2.0.0 Tests if two 3D geometries are entirely within a given 3D distance
- **ST\_3DDWithin** - Availability: 2.0.0 Tests if two 3D geometries are within a given 3D distance
- **ST\_3DDistance** - Disponibilidad: 2.0.0 Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units.
- **ST\_3DIntersects** - Availability: 2.0.0 Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- **ST\_3DLongestLine** - Disponibilidad: 2.0.0 Returns the 3D longest line between two geometries
- **ST\_3DMaxDistance** - Disponibilidad: 2.0.0 Returns the 3D cartesian maximum distance (based on spatial ref) between two geometries in projected units.
- **ST\_3DShortestLine** - Disponibilidad: 2.0.0 Returns the 3D shortest line between two geometries

- **ST\_AsLatLonText** - Disponibilidad: 2.0 Return the Degrees, Minutes, Seconds representation of the given point.
- **ST\_AsX3D** - Availability: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML Returns a Geometry in X3D xml node element format: ISO-IEC-19776-1.2-X3DEncodings-XML
- **ST\_CollectionHomogenize** - Disponibilidad: 2.0.0 Returns the simplest representation of a geometry collection.
- **ST\_ConcaveHull** - Disponibilidad: 2.0.0 Computes a possibly concave geometry that contains all input geometry vertices
- **ST\_FlipCoordinates** - Disponibilidad: 2.0.0 Returns a version of a geometry with X and Y axis flipped.
- **ST\_GeomFromGeoJSON** - Disponibilidad: 2.0.0 necesita de - JSON-C >= 0.9 Toma como entrada una representación geojson de una geometría y devuelve un objeto geométrico PostGIS
- **ST\_InterpolatePoint** - Disponibilidad: 2.0.0 Devuelve el valor de la dimensión medida de una geometría en el punto cerrado al punto proporcionado.
- **ST\_IsValidDetail** - Availability: 2.0.0 Returns a valid\_detail row stating if a geometry is valid or if not a reason and a location.
- **ST\_IsValidReason** - Availability: 2.0 version taking flags. Returns text stating if a geometry is valid, or a reason for invalidity.
- **ST\_MakeLine** - Disponibilidad: 2.0.0 - Se introdujo el soporte de una cadena lineal como elemento de entrada Crea una cadena de línea desde geometrías de punto, multipunto o de línea.
- **ST\_MakeValid** - Availability: 2.0.0 Attempts to make an invalid geometry valid without losing vertices.
- **ST\_Node** - Availability: 2.0.0 Nodes a collection of lines.
- **ST\_NumPatches** - Disponibilidad: 2.0.0 Devuelve el número de caras en una superficie poliédrica. Devolverá nulo para geometrías no poliédricas.
- **ST\_OffsetCurve** - Disponibilidad: 2.0 Returns an offset line at a given distance and side from an input line.
- **ST\_PatchN** - Disponibilidad: 2.0.0 Devuelve el tipo de geometría del valor de ST\_Geometry.
- **ST\_Perimeter** - Availability 2.0.0: Support for geography was introduced Returns the length of the boundary of a polygonal geometry or geography.
- **ST\_Project** - Disponibilidad: 2.0.0 Returns a point projected from a start point by a distance and bearing (azimuth).
- **ST\_RelateMatch** - Availability: 2.0.0 Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern
- **ST\_SharedPaths** - Disponibilidad: 2.0.0 Returns a collection containing paths shared by the two input linestrings/multi-linestrings.
- **ST\_Snap** - Disponibilidad: 2.0.0 Ajusta segmentos y vértices de la geometría de entrada a vértices de una geometría de referencia.
- **ST\_Split** - Availability: 2.0.0 requires GEOS Returns a collection of geometries created by splitting a geometry by another geometry.
- **ST\_UnaryUnion** - Availability: 2.0.0 Computes the union of the components of a single geometry.

#### Functions enhanced in PostGIS 2.0

- **&&** - Mejorado: 2.0.0 soporte para superficies poliédricas fue introducida. Returns TRUE if A's 2D bounding box intersects B's 2D bounding box.
- **AddGeometryColumn** - Mejorada: 2.0.0 introducción del argumento use\_typpmod. El valor predeterminado es crear columnas de geometrías basadas en typpmod en lugar de las basadas en restricciones. Suprime una columna de geometrías de una tabla espacial.
- **Box2D** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Returns a BOX2D representing the 2D extent of a geometry.

- **Box3D** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Returns a BOX3D representing the 3D extent of a geometry.
  - **GeometryType** - Mejorado: 2.0.0 se introdujo soporte para superficies poliédricas, Triangulos y TIN. Devuelve el tipo de geometría del valor de ST\_Geometry.
  - **Populate\_Geometry\_Columns** - Mejorado: 2.0.0 el argumento opcional use\_tymod fue introducido y permite controlar si las columnas se crean con modificadores de tipo o con restricciones de tipo check. Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
  - **ST\_3DExtent** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Aggregate function that returns the 3D bounding box of geometries.
  - **ST\_Affine** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Apply a 3D affine transformation to a geometry.
  - **ST\_Area** - Enhanced: 2.0.0 - support for 2D polyhedral surfaces was introduced. Returns the area of a polygonal geometry.
  - **ST\_AsBinary** - Mejorado: 2.0.0 soporte para superficies poliédricas, triángulos y TIN fue introducida. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
  - **ST\_AsBinary** - Enhanced: 2.0.0 support for higher coordinate dimensions was introduced. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
  - **ST\_AsBinary** - Enhanced: 2.0.0 support for specifying endian with geography was introduced. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
  - **ST\_AsEWKB** - Mejorado: 2.0.0 soporte para superficies poliédricas, triángulos y TIN fue introducida. Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
  - **ST\_AsEWKT** - Enhanced: 2.0.0 support for Geography, Polyhedral surfaces, Triangles and TIN was introduced. Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.
  - **ST\_AsGML** - Enhanced: 2.0.0 prefix support was introduced. Option 4 for GML3 was introduced to allow using LineString instead of Curve tag for lines. GML3 Support for Polyhedral surfaces and TINS was introduced. Option 32 was introduced to output the box. Return the geometry as a GML version 2 or 3 element.
  - **ST\_AsKML** - Enhanced: 2.0.0 - Add prefix namespace, use default and named args Return the geometry as a KML element.
  - **ST\_Azimuth** - Enhanced: 2.0.0 support for geography was introduced. Returns the north-based azimuth of a line between two points.
  - **ST\_Dimension** - Mejora: 2.0.0 se introdujeron soporte de superficies poliédricas y TIN. No lanza una excepción si se envía una geometría vacía. Devuelve la dimensión de las coordenadas del valor de ST\_Geometry.
  - **ST\_Dump** - Mejorado: 2.0.0 se introdujo soporte para superficies poliédricas, Triangulos y TIN. Returns a set of geometry\_dump rows for the components of a geometry.
  - **ST\_Expand** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Returns a bounding box expanded from another bounding box or a geometry.
  - **ST\_Extent** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Aggregate function that returns the bounding box of geometries.
  - **ST\_Force2D** - Mejorado: 2.0.0 soporte para superficies poliédricas fue introducida. Forzar las geometrías en un "modo de 2 dimensiones".
  - **ST\_Force3D** - Mejorado: 2.0.0 soporte para superficies poliédricas fue introducida. Forzar las geometrías en modo XYZ. Este es un alias para ST\_Force3DZ.
  - **ST\_Force3DZ** - Mejorado: 2.0.0 soporte para superficies poliédricas fue introducida. Fuerza las geometrías en modo XYZ.
  - **ST\_ForceCollection** - Mejorado: 2.0.0 soporte para superficies poliédricas fue introducida. Convertir la geometría en una GEOMETRYCOLLECTION.
-



- **ST\_ForceRHR** - Mejorado: 2.0.0 soporte para superficies poliédricas fue introducida. Fuerza la orientación de los vértices en un polígono para seguir la regla de la mano derecha.
  - **ST\_GMLToSQL** - Mejora: 2.0.0 se introdujeron soporte de superficies poliédricas y TIN. Devuelve un valor específico ST\_Geometry desde una representación GML. Esto es un alias de ST\_GeomFromGML
  - **ST\_GMLToSQL** - Mejorada: 2.0.0 se agregó el parámetro por defecto opcional srid. Devuelve un valor específico ST\_Geometry desde una representación GML. Esto es un alias de ST\_GeomFromGML
  - **ST\_GeomFromEWKB** - Mejora: 2.0.0 se introdujeron soporte de superficies poliédricas y TIN. Devuelve un valor específico de ST\_Geometry desde una representación "Extended Well-Known Binary" (EWKB).
  - **ST\_GeomFromEWKT** - Mejora: 2.0.0 se introdujeron soporte de superficies poliédricas y TIN. Devuelve un valor especificado ST\_Geometry desde una representación "Extended Well-Known Text" (EWKT).
  - **ST\_GeomFromGML** - Mejora: 2.0.0 se introdujeron soporte de superficies poliédricas y TIN. Toma una representación GML como entrada de una geometría y extrae un objeto geométrico PostGIS
  - **ST\_GeomFromGML** - Mejorada: 2.0.0 se agregó el parámetro por defecto opcional srid. Toma una representación GML como entrada de una geometría y extrae un objeto geométrico PostGIS
  - **ST\_GeometryN** - Mejorado: 2.0.0 se introdujo soporte para superficies poliédricas, Triangulos y TIN. Devuelve el tipo de geometría del valor de ST\_Geometry.
  - **ST\_GeometryType** - Mejora: 2.0.0 se introdujo soporte de superficies poliédricas. Devuelve el tipo de geometría del valor de ST\_Geometry.
  - **ST\_IsClosed** - Mejora: 2.0.0 se introdujo soporte de superficies poliédricas. Devuelve TRUE si los puntos de inicio y final de una LINESTRING son coincidentes. Para superficies poliedricas si son cerradas (volumetricas).
  - **ST\_MakeEnvelope** - Mejorado: 2.0: Se introdujo capacidad de especificar una caja sin especificar un SRID. Crea un polígono rectangular formado a partir de los mínimos y máximos especificados. Los valores de entrada deben estar en el SRS especificado en el SRID.
  - **ST\_MakeValid** - Enhanced: 2.0.1, speed improvements Attempts to make an invalid geometry valid without losing vertices.
  - **ST\_NPoints** - Mejora: 2.0.0 se introdujo soporte de superficies poliédricas. Devuelve el numero de puntos (vértices) en la geometría.
  - **ST\_NumGeometries** - Mejorado: 2.0.0 se introdujo soporte para superficies poliédricas, Triangulos y TIN. Devuelve el numero de puntos en la geometría. Funciona con todas las geometrías.
  - **ST\_NumPoints** - Mejorado: 2.0.0 se introdujo soporte para superficies poliédricas, Triangulos y TIN. Devuelve un resumen de texto del contenido de la geometría.
  - **ST\_Relate** - Enhanced: 2.0.0 - added support for specifying boundary node rule. Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix
  - **ST\_Rotate** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Rotates a geometry about an origin point.
  - **ST\_Rotate** - Enhanced: 2.0.0 additional parameters for specifying the origin of rotation were added. Rotates a geometry about an origin point.
  - **ST\_RotateX** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Rotates a geometry about the X axis.
  - **ST\_RotateY** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Rotates a geometry about the Y axis.
  - **ST\_RotateZ** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Rotates a geometry about the Z axis.
  - **ST\_Scale** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Scales a geometry by given factors.
-

- **ST\_ShiftLongitude** - Mejora: 2.0.0 se introdujeron soporte de superficies poliédricas y TIN. Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- **ST\_Summary** - Mejorado: 2.0.0 agregó soporte para geography Devuelve un resumen de texto del contenido de la geometría.
- **ST\_Transform** - Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Return a new geometry with coordinates transformed to a different spatial reference system.

#### Functions changed in PostGIS 2.0

- **AddGeometryColumn** - Cambiado: 2.0.0 Esta función ya no se actualiza desde geometry\_columns ya que geometry\_columns es una vista que se lee desde los catálogos del sistema. Por defecto tampoco crea las restricciones, sino que utiliza el modificador de tipo de PostgreSQL. Así que para la construcción de una columna de tipo POINT en wgs84 con esta función ejemplo que hoy es equivalente a: ALTER TABLE some\_table ADD COLUMN geom geometry(Point,4326); Suprime una columna de geometrías de una tabla espacial.
- **AddGeometryColumn** - Cambiado: 2.0.0 Si necesitas el comportamiento antiguo de restricciones, utiliza el valor predeterminado use\_typmod, pero cambiala a false. Suprime una columna de geometrías de una tabla espacial.
- **AddGeometryColumn** - Cambiado: 2.0.0 Las Vistas ya no pueden ser registradas manualmente en geometry\_columns, no obstante las vistas se que construyan a partir de geometrías typmod de las tablas de geometrías y sean utilizadas sin funciones wrapper se registraran correctamente porque heredan el comportamiento typmod de su columna de la tabla padre. Las vistas que utilizan funciones de geometría que devuelvan geometrías necesitarán de transformación cast a geometrías typmod para esta columnas de geometrías de la vista y que se registren correctamente en geometry\_columns. Consulta . Suprime una columna de geometrías de una tabla espacial.
- **DropGeometryColumn** - Cambiado: 2.0.0 Se proporciona esta función para la compatibilidad con versiones anteriores. Ahora que geometry\_columns es una vista y no un catálogo del sistema, se puede eliminar una columna de geometría como cualquier otra columna de la tabla utilizando ALTER TABLE Suprime una columna de geometrías de una tabla espacial.
- **DropGeometryTable** - Cambiado: 2.0.0 Se proporciona esta función para la compatibilidad con versiones anteriores. Ahora que geometry\_columns es una vista y no un catálogo del sistema, se puede borrar una tabla con columnas de geometría como cualquier otra tabla utilizando DROP TABLE Borra una tabla y todas sus referencias en la tabla geométrica\_columns.
- **Populate\_Geometry\_Columns** - Cambiado: 2.0.0 Por defecto, ahora utiliza modificadores de tipo en lugar de restricciones de tipo check para limitar los tipos de geometría. Puedes seguir utilizando el comportamiento de las restricciones check con el uso de la nueva variable use\_typmod y estableciéndolo a false. Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- **ST\_3DExtent** - Changed: 2.0.0 In prior versions this used to be called ST\_Extent3D Aggregate function that returns the 3D bounding box of geometries.
- **ST\_3DLength** - Changed: 2.0.0 In prior versions this used to be called ST\_Length3D Returns the 3D length of a linear geometry.
- **ST\_3DMakeBox** - Changed: 2.0.0 In prior versions this used to be called ST\_MakeBox3D Creates a BOX3D defined by two 3D point geometries.
- **ST\_3DPerimeter** - Changed: 2.0.0 In prior versions this used to be called ST\_Perimeter3D Returns the 3D perimeter of a polygonal geometry.
- **ST\_AsBinary** - Changed: 2.0.0 Inputs to this function can not be unknown -- must be geometry. Constructs such as ST\_AsBinary('POINT(1 2)') are no longer valid and you will get an n\_st\_asbinary(unknown) is not unique error. Code like that needs to be changed to ST\_AsBinary('POINT(1 2)::geometry');. If that is not possible, then install legacy.sql. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsGML** - Changed: 2.0.0 use default named args Return the geometry as a GML version 2 or 3 element.
- **ST\_AsGeoJSON** - Changed: 2.0.0 support default args and named args. Return a geometry as a GeoJSON element.
- **ST\_AsSVG** - Changed: 2.0.0 to use default args and support named args Returns SVG path data for a geometry.



- **ST\_EndPoint** - Cambiado: 2.0.0 ya no funciona con multilinestrings de geometrías simples. En versiones anteriores de PostGIS -- una línea simple multilinestring funciona sin problemas con esta función y devuelve el punto inicial. En la versión 2.0.0 simplemente devuelve NULL como con cualquier multilinestring. La antigua versión era una función sin documentar, pero la gente que asumía que tenía sus datos almacenados en LINESTRING pueden experimentar este comportamiento ahora de resultado NULL en la versión 2.0. Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString.
- **ST\_GeomFromText** - Cambiado: 2.0.0 En las versiones anteriores de PostGIS ST\_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') estaba permitido. Esto no está permitido ahora en PostGIS 2.0.0 para ajustarse mejor a las normas SQL/MM. Esto debería ser escrito como ST\_GeomFromText('GEOMETRYCOLLECTION EMPTY') Devuelve un valor específico de ST\_Geometry desde una representación "Extended Well-Known Binary" (EWKB).
- **ST\_GeometryN** - Cambiado: 2.0.0 Versiones anteriores devuelven NULL para geometrías simples. Esto ha sido cambiado para devolver la geometría en el caso de ST\_GeometryN(..,1) . Devuelve el tipo de geometría del valor de ST\_Geometry.
- **ST\_IsEmpty** - Cambiado: 2.0.0 En las versiones anteriores de PostGIS ST\_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') estaba permitido. Esto no está permitido ahora en PostGIS 2.0.0 para ajustarse mejor a las normas SQL/MM. Tests if a geometry is empty.
- **ST\_Length** - Changed: 2.0.0 Breaking change -- in prior versions applying this to a MULTI/POLYGON of type geography would give you the perimeter of the POLYGON/MULTIPOLYGON. In 2.0.0 this was changed to return 0 to be in line with geometry behavior. Please use ST\_Perimeter if you want the perimeter of a polygon Returns the 2D length of a linear geometry.
- **ST\_LocateAlong** - Modificado: 2.0.0 en versiones anteriores éste solía llamarse ST\_Locate\_Along\_Measure. El nombre anterior ha quedado obsoleto y se eliminará en el futuro, pero aún está disponible. Returns the point(s) on a geometry that match a measure value.
- **ST\_LocateBetween** - Modificado: 2.0.0 en versiones anteriores éste solía llamarse ST\_Locate\_Along\_Measure. El nombre anterior ha quedado obsoleto y se eliminará en el futuro, pero aún está disponible. Returns the portions of a geometry that match a measure range.
- **ST\_NumGeometries** - Cambiado: 2.0.0 En versiones anteriores esto devolvería NULL si la geometría no era de tipo collection/MULTI. 2.0.0+ devuelve 1 para geometrías simples, por ejemplo, POLYGON, LINESTRING, POINT. Devuelve el número de puntos en la geometría. Funciona con todas las geometrías.
- **ST\_NumInteriorRings** - Cambiado: 2.0.0 - En versiones anteriores permitiría pasar un multipolígono, devolviendo el número de anillos interiores de primer polígono. Devuelva el número de anillos interiores de una geometría poligonal.
- **ST\_PointN** - Cambiado: 2.0.0 ya no funciona con una sola geometría multilinestrings. En versiones antiguas de PostGIS -- una sola línea MultiLineString trabajaría felizmente con esta función y regresaría el punto de inicio. En 2.0.0 sólo devuelve NULL como cualquier otro MultiLineString. Devuelve el número de puntos en un valor ST\_LineString o ST\_CircularString.
- **ST\_StartPoint** - Cambiado: 2.0.0 ya no funciona con multilinestrings de geometrías simples. En versiones anteriores de PostGIS -- una línea simple multilinestring funciona sin problemas con esta función y devuelve el punto inicial. En la versión 2.0.0 simplemente devuelve NULL como con cualquier multilinestring. La antigua versión era una función sin documentar, pero la gente que asumía que tenía sus datos almacenados en LINESTRING pueden experimentar este comportamiento ahora de resultado NULL en la versión 2.0. Returns the first point of a LineString.

### 12.12.12 PostGIS Functions new or enhanced in 1.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.5

- **&&** - Availability: 1.5.0 support for geography was introduced. Returns TRUE if A's 2D bounding box intersects B's 2D bounding box.
- **PostGIS\_LibXML\_Version** - Availability: 1.5 Returns the version number of the libxml2 library.
- **ST\_AddMeasure** - Disponibilidad: 1.5.0 Interpolates measures along a linear geometry.

- **ST\_AsBinary** - Availability: 1.5.0 geography support was introduced. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
  - **ST\_AsGML** - Availability: 1.5.0 geography support was introduced. Return the geometry as a GML version 2 or 3 element.
  - **ST\_AsGeoJSON** - Availability: 1.5.0 geography support was introduced. Return a geometry as a GeoJSON element.
  - **ST\_AsText** - Availability: 1.5 - support for geography was introduced. Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.
  - **ST\_Buffer** - Availability: 1.5 - ST\_Buffer was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added. Computes a geometry covering all points within a given distance from a geometry.
  - **ST\_ClosestPoint** - Disponibilidad: 1.5.0 Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.
  - **ST\_CollectionExtract** - Disponibilidad: 1.5.0 Given a geometry collection, returns a multi-geometry containing only elements of a specified type.
  - **ST\_Covers** - Availability: 1.5 - support for geography was introduced. Tests if every point of B lies in A
  - **ST\_DFullyWithin** - Availability: 1.5.0 Tests if two geometries are entirely within a given distance
  - **ST\_DWithin** - Availability: 1.5.0 support for geography was introduced Tests if two geometries are within a given distance
  - **ST\_Distance** - Availability: 1.5.0 geography support was introduced in 1.5. Speed improvements for planar to better handle large or many vertex geometries Returns the distance between two geometry or geography values.
  - **ST\_DistanceSphere** - Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points. Returns minimum distance in meters between two lon/lat geometries using a spherical earth model.
  - **ST\_DistanceSpheroid** - Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points. Returns the minimum distance between two lon/lat geometries using a spheroidal earth model.
  - **ST\_Envelope** - Disponibilidad: 1.5.0 comportamiento modificado para devolver doble precisión en vez de float4. Devuelve una geometría que representa la caja en doble precisión (float8) de la geometría dada.
  - **ST\_Expand** - Availability: 1.5.0 behavior changed to output double precision instead of float4 coordinates. Returns a bounding box expanded from another bounding box or a geometry.
  - **ST\_GMLToSQL** - Disponibilidad: 1.5, requiere libxml2 1.6+ Devuelve un valor específico ST\_Geometry desde una representación GML. Esto es un alias de ST\_GeomFromGML
  - **ST\_GeomFromGML** - Disponibilidad: 1.5, requiere libxml2 1.6+ Toma una representación GML como entrada de una geometría y extrae un objeto geométrico PostGIS
  - **ST\_GeomFromKML** - Availability: 1.5, requiere libxml2 2.6+ Toma una representación de una geometría KML de entrada y devuelve un objeto geométrico PostGIS
  - **ST\_HausdorffDistance** - Disponibilidad: 1.5.0 Returns the Hausdorff distance between two geometries.
  - **ST\_Intersection** - Availability: 1.5 support for geography data type was introduced. Computes a geometry representing the shared portion of geometries A and B.
  - **ST\_Intersects** - Availability: 1.5 support for geography was introduced. Tests if two geometries intersect (they have at least one point in common)
  - **ST\_Length** - Availability: 1.5.0 geography support was introduced in 1.5. Returns the 2D length of a linear geometry.
  - **ST\_LongestLine** - Disponibilidad: 1.5.0 Returns the 2D longest line between two geometries.
  - **ST\_MakeEnvelope** - Disponibilidad: 1.5 Crea un polígono rectangular formado a partir de los mínimos y máximos especificados. Los valores de entrada deben estar en el SRS especificado en el SRID.
-

- **ST\_MaxDistance** - Disponibilidad: 1.5.0 Returns the 2D largest distance between two geometries in projected units.
- **ST\_NumPoints** - Disponibilidad: 1.2.2 Devuelve un resumen de texto del contenido de la geometría.
- **ST\_ShortestLine** - Disponibilidad: 1.5.0 Returns the 2D shortest line between two geometries
- **~=** - Availability: 1.5.0 changed behavior Returns TRUE if A's bounding box is the same as B's.

### 12.12.13 PostGIS Functions new or enhanced in 1.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.4

- **Populate\_Geometry\_Columns** - Disponibilidad: 1.4.0 Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- **ST\_ContainsProperly** - Availability: 1.4.0 Tests if every point of B lies in the interior of A
- **ST\_GeoHash** - Disponibilidad: 1.4.0 Return a GeoHash representation of the geometry.
- **ST\_GeomCollFromText** - Disponibilidad: 1.4.0 - **ST\_MakeLine** (geomarray) fue introducido. Las Funciones agregadas **ST\_MakeLine** se mejoraron para manejar más puntos más rápido. Creates a GeometryCollection or Multi\* geometry from a set of geometries.
- **ST\_IsValidReason** - Availability: 1.4 Returns text stating if a geometry is valid, or a reason for invalidity.
- **ST\_LineCrossingDirection** - Availability: 1.4 Returns a number indicating the crossing behavior of two LineStrings
- **ST\_LocateBetweenElevations** - Disponibilidad: 1.4.0 Returns the portions of a geometry that lie in an elevation (Z) range.
- **ST\_MakeLine** - Disponibilidad: 1.4.0 - **ST\_MakeLine** (geomarray) fue introducido. Las Funciones agregadas **ST\_MakeLine** se mejoraron para manejar más puntos más rápido. Crea una cadena de línea desde geometrías de punto, multipunto o de línea.
- **ST\_MinimumBoundingCircle** - Disponibilidad: 1.4.0 Returns the smallest circle polygon that contains a geometry.
- **ST\_Union** - Availability: 1.4.0 - **ST\_Union** was enhanced. **ST\_Union**(geomarray) was introduced and also faster aggregate collection in PostgreSQL. Computes a geometry representing the point-set union of the input geometries.

### 12.12.14 PostGIS Functions new or enhanced in 1.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.3

- **ST\_AsGML** - Disponibilidad: 1.3.2 Return the geometry as a GML version 2 or 3 element.
- **ST\_AsGeoJSON** - Disponibilidad: 1.3.4 Return a geometry as a GeoJSON element.
- **ST\_CurveToLine** - Availability: 1.3.0 Converts a geometry containing curves to a linear geometry.
- **ST\_LineToCurve** - Availability: 1.3.0 Converts a linear geometry to a curved geometry.
- **ST\_SimplifyPreserveTopology** - Disponibilidad: 1.3.3 Returns a simplified and valid version of a geometry, using the Douglas-Peucker algorithm.

## Chapter 13

# Informar de problemas

### 13.1 Informar sobre errores de software

Informar sobre errores efectivamente es una manera fundamental de ayudar en el desarrollo de PostGIS. El informe de errores más efectivo es el que permite a los desarrolladores de PostGIS reproducirlo, así que lo ideal sería que contenga un trozo de código que lo genere y toda la información posible del entorno en el que fue detectado. Una información bastante buena se puede obtener ejecutando `SELECT postgis_full_version()` [para postgis] y `SELECT version()` [para postgresql].

Si no está usando la última versión, vale la pena echar un vistazo a su [lista de cambios en la versión](#) primero, para ver si el error ya se ha solucionado.

Usar el [seguimiento de errores de PostGIS](#) servirá para asegurarnos de que nuestros informes no son descartados, y nos mantendrá informados de como progresa su gestión. Antes de informar acerca de un nuevo error por favor consulte la base de datos para ver si es uno ya conocido, y si lo es, por favor, agregue la nueva información que tenga sobre él.

Puede leer la documentación de Simon Tatham acerca de [Cómo informar de errores de manera eficiente](#) antes de rellenar un nuevo informe.

### 13.2 Informando sobre problemas de documentación

La documentación debería reflejar con precisión las características y comportamiento del software. Si no es así, podría deberse a un error del software, o porque la documentación es deficiente o errónea.

Los problemas con la documentación se pueden enviar también al [Seguimiento de errores de PostGIS](#).

Si su revisión es de poca importancia, tan sólo descríbala en un nuevo asunto de la lista de seguimiento, y sea específico acerca de en que parte de la documentación se encuentra.

Si sus cambios son más extensos, es preferible un envío por Subversion. Este es un proceso en cuatro pasos en Unix (suponiendo que tiene instalado ya [Subversion](#)):

1. Consiga una copia de la rama de PostGIS en Subversion. En Unix, teclee:

```
git clone https://git.osgeo.org/gitea/postgis/postgis.git
```

Ésta se almacenará en la carpeta `./trunk`

2. Haga los cambios a la documentación con su editor favorito. En Unix, teclee (por ejemplo):

```
vim trunk/doc/postgis.xml
```

Tenga en cuenta que la documentación está escrita en DocBook XML y no en HTML, así que si no está familiarizado con este formato por favor siga el ejemplo del resto de la documentación.

---

3. Haga un fichero para solucionarlo que contenga las diferencias desde la copia maestra de la documentación. En Unix, teclee:

**svn diff trunk/doc/postgis.xml > doc.patch**

4. Adjunte la solución a un nuevo tema en el seguimiento de errores.

## Appendix A

## Apéndice

### A.1 PostGIS 3.4.0beta1

2023/07/14

This version requires PostgreSQL 12 or higher, GEOS 3.6 or higher, and Proj 6.1+. To take advantage of all features, GEOS 3.12+ is needed. To take advantage of all SFCGAL features, SFCGAL 1.4.1+ is needed.

NOTE: GEOS 3.12.0 details at [GEOS 3.12.0 release notes](#)

Many thanks to our translation teams, in particular:

Teramoto Ikuhiro (Japanese Team)

Vincent Bre (French Team)

#### A.1.1 New features

[5055](#), complete manual internationalization (Sandro Santilli)

[5052](#), target version support in `postgis_extensions_upgrade` (Sandro Santilli)

[5306](#), expose version of GEOS at compile time (Sandro Santilli)

New `install-extension-upgrades` command in `postgis` script (Sandro Santilli)

[5257](#), [5261](#), [5277](#), Support changes for PostgreSQL 16 (Regina Obe)

[5006](#), [705](#), `ST_Transform`: Support PROJ pipelines (Robert Coup, Koordinates)

[5283](#), `[postgis_topology] RenameTopology` (Sandro Santilli)

[5286](#), `[postgis_topology] RenameTopoGeometryColumn` (Sandro Santilli)

[703](#), `[postgis_raster]` Add min/max resampling as options (Christian Schroeder)

[5336](#), `[postgis_topology]` `topogeometry` cast to `topelement` support (Regina Obe)

Allow singleton geometry to be inserted into `Geometry(Multi*)` columns (Paul Ramsey)

[721](#), New window-based `ST_ClusterWithinWin` and `ST_ClusterIntersectingWin` (Paul Ramsey)

[5397](#), `[address_standardizer]` `debug_standardize_address` function (Regina Obe)

[5373](#) `ST_LargestEmptyCircle`, exposes extra semantics on circle finding. Geos 3.9+ required (Martin Davis)

[5267](#), `ST_Project` signature for geometry, and two-point signature (Paul Ramsey)

[5267](#), `ST_LineExtend` for extending linestrings (Paul Ramsey)

---

### A.1.2 Mejoras

5194, do not update system catalogs from postgis\_extensions\_upgrade (Sandro Santilli)

5092, reduce number of upgrade paths installed on system (Sandro Santilli)

635, honour --bindir (and --prefix) configure switch for executables (Sandro Santilli)

Honour --mandir (and --prefix) configure switch for man pages install path (Sandro Santilli)

Honour --htmldir (and --docdir and --prefix) configure switch for html pages install path (Sandro Santilli)

[postgis\_topology] Speed up check of topology faces without edges (Sandro Santilli)

[postgis\_topology] Speed up coincident nodes check in topology validation (Sandro Santilli)

718, ST\_QuantizeCoordinates(): speed-up implementation (Even Rouault)

Repair spatial planner stats to use computed selectivity for contains/within queries (Paul Ramsey)

734, Additional metadata on Proj installation in postgis\_proj\_version (Paul Ramsey)

5177, allow building tools without PostgreSQL server headers (Sandro Santilli)

ST\_Project signature for geometry, and two-point signature (Paul Ramsey)

4913, ST\_AsSVG support for curve types CircularString, CompoundCurve, MultiCurve, and MultiSurface (Regina Obe)

5266, ST\_ClosestPoint, ST\_ShortestLine support for geography type (MobilityDB Esteban Zimanyi, Paul Ramsey)

### A.1.3 Breaking Changes

5229, Drop support for Proj < 6.1 and PG 11 (Regina Obe)

5306, 734, postgis\_full\_version() and postgis\_proj\_version() now output more information about proj network configuration and data paths. GEOS compile-time version also shown if different from run-time (Paul Ramsey, Sandro Santilli)