

PostGIS 3.6.1dev ☒☒☒☒☒☒

**DEV (Tue 02 Sep 2025 01:58:28 AM UTC
rev. 0f6b4da)**

Contents

1	简介	1
1.1	安装与升级	1
1.2	许可证 - 简介	2
1.3	许可证 - 详细	2
1.4	许可证 - 附录	3
2	PostGIS 简介	6
2.1	简介	6
2.2	安装与升级	6
2.2.1	简介	7
2.2.2	安装与升级	7
2.2.3	简介	8
2.2.4	简介	10
2.2.5	PostGIS Extensions 简介	10
2.2.6	简介	12
2.2.7	简介	15
2.3	安装与升级	16
2.4	Installing, Upgrading Tiger Geocoder, and loading data	16
2.4.1	Tiger Geocoder Enabling your PostGIS database	17
2.4.2	安装与升级 TIGER 数据	19
2.4.3	Required tools for tiger data loading	19
2.4.4	Upgrading your Tiger Geocoder Install and Data	20
2.5	安装与升级	20
3	PostGIS Administration	22
3.1	Performance Tuning	22
3.1.1	Startup	22
3.1.2	Runtime	23
3.2	Configuring raster support	23
3.3	安装与升级	24

3.3.1	Spatially enable database using EXTENSION	24
3.3.2	Spatially enable database without using EXTENSION (discouraged)	24
3.4	Upgrading spatial databases	25
3.4.1	Soft upgrade	25
3.4.1.1	Soft Upgrade 9.1+ using extensions	25
3.4.1.2	Soft Upgrade Pre 9.1+ or without extensions	26
3.4.2	Hard upgrade	27

4 Data Management 29

4.1	GIS (OGC) Geometries	29
4.1.1	OGC Geometry	29
4.1.1.1	Point	30
4.1.1.2	LineString	30
4.1.1.3	LinearRing	30
4.1.1.4	Polygon	30
4.1.1.5	MultiPoint	30
4.1.1.6	MultiLineString	31
4.1.1.7	MultiPolygon	31
4.1.1.8	GeometryCollection	31
4.1.1.9	PolyhedralSurface	31
4.1.1.10	Triangle	31
4.1.1.11	TIN	31
4.1.2	SQL-MM Part 3	32
4.1.2.1	CircularString	32
4.1.2.2	CompoundCurve	32
4.1.2.3	CurvePolygon	32
4.1.2.4	MultiCurve	33
4.1.2.5	MultiSurface	33
4.1.3	OpenGIS WKB ↔ WKT	33
4.2	Geometry Data Type	34
4.2.1	OpenGIS WKB ↔ WKT	34
4.3	PostGIS Geometries	36
4.3.1	Geometries	37
4.3.2	PostGIS Geometries	38
4.3.3	Geometries and the Geometry Column	39
4.3.4	Geometries FAQ	39
4.4	Geometry Validation	39
4.4.1	Simple Geometry	40
4.4.2	Valid Geometry	42

4.4.3	Managing Validity	44
4.5	SPATIAL_REF_SYS 简体中文版	45
4.5.1	SPATIAL_REF_SYS Table	45
4.5.2	SPATIAL_REF_SYS 简体中文版	46
4.6	简体中文版	47
4.6.1	简体中文版	47
4.6.2	The GEOMETRY_COLUMNS VIEW	48
4.6.3	geometry_columns 简体中文版	48
4.7	GIS (GIS) 简体中文版	50
4.7.1	SQL 简体中文版	51
4.7.2	shp2pgsql: ESRI shapefile 简体中文版	51
4.8	简体中文版	53
4.8.1	SQL 简体中文版	53
4.8.2	简体中文版	54
4.9	简体中文版	54
4.9.1	GiST 简体中文版	55
4.9.2	GiST 简体中文版	55
4.9.3	GiST 简体中文版	57
4.9.4	简体中文版	58
5	Spatial Queries	59
5.1	Determining Spatial Relationships	59
5.1.1	Dimensionally Extended 9-Intersection Model	59
5.1.2	Named Spatial Relationships	61
5.1.3	General Spatial Relationships	62
5.2	Using Spatial Indexes	64
5.3	Examples of Spatial SQL	65
6	简体中文版	68
6.1	简体中文版	68
6.1.1	简体中文版	68
6.1.2	简体中文版	68
6.2	简体中文版	69
6.3	简体中文版	69

7	PostGIS Reference	70
7.1	PostgreSQL PostGIS Geometry/Geography/Box 表表	70
7.1.1	box2d	70
7.1.2	box3d	71
7.1.3	geometry	71
7.1.4	geometry_dump	72
7.1.5	geography	72
7.2	表表表表	73
7.2.1	AddGeometryColumn	73
7.2.2	DropGeometryColumn	75
7.2.3	DropGeometryTable	75
7.2.4	Find_SRID	76
7.2.5	Populate_Geometry_Columns	77
7.2.6	UpdateGeometrySRID	78
7.3	表表表表表 (constructor)	79
7.3.1	ST_Collect	79
7.3.2	ST_LineFromMultiPoint	81
7.3.3	ST_MakeEnvelope	82
7.3.4	ST_MakeLine	82
7.3.5	ST_MakePoint	84
7.3.6	ST_MakePointM	85
7.3.7	ST_MakePolygon	87
7.3.8	ST_Point	88
7.3.9	ST_PointZ	90
7.3.10	ST_PointM	90
7.3.11	ST_PointZM	91
7.3.12	ST_Polygon	91
7.3.13	ST_TileEnvelope	92
7.3.14	ST_HexagonGrid	93
7.3.15	ST_Hexagon	96
7.3.16	ST_SquareGrid	97
7.3.17	ST_Square	98
7.3.18	ST_Letters	99
7.4	表表表表表 (accessor)	100
7.4.1	GeometryType	100
7.4.2	ST_Boundary	102
7.4.3	ST_BoundingDiagonal	104
7.4.4	ST_CoordDim	105
7.4.5	ST_Dimension	105

7.4.6 ST_Dump	106
7.4.7 ST_DumpPoints	108
7.4.8 ST_DumpSegments	112
7.4.9 ST_DumpRings	114
7.4.10 ST_EndPoint	115
7.4.11 ST_Envelope	116
7.4.12 ST_ExteriorRing	118
7.4.13 ST_GeometryN	119
7.4.14 ST_GeometryType	121
7.4.15 ST_HasArc	122
7.4.16 ST_InteriorRingN	123
7.4.17 ST_NumCurves	124
7.4.18 ST_CurveN	124
7.4.19 ST_IsClosed	125
7.4.20 ST_IsCollection	127
7.4.21 ST_IsEmpty	128
7.4.22 ST_IsPolygonCCW	129
7.4.23 ST_IsPolygonCW	130
7.4.24 ST_IsRing	131
7.4.25 ST_IsSimple	131
7.4.26 ST_M	132
7.4.27 ST_MemSize	133
7.4.28 ST_NDims	134
7.4.29 ST_NPoints	135
7.4.30 ST_NRings	136
7.4.31 ST_NumGeometries	136
7.4.32 ST_NumInteriorRings	137
7.4.33 ST_NumInteriorRing	138
7.4.34 ST_NumPatches	138
7.4.35 ST_NumPoints	139
7.4.36 ST_PatchN	140
7.4.37 ST_PointN	141
7.4.38 ST_Points	142
7.4.39 ST_StartPoint	143
7.4.40 ST_Summary	144
7.4.41 ST_X	145
7.4.42 ST_Y	146
7.4.43 ST_Z	147
7.4.44 ST_Zmflag	148

7.4.45	ST_HasZ	148
7.4.46	ST_HasM	149
7.5	国国国国国 (editor)	150
7.5.1	ST_AddPoint	150
7.5.2	ST_CollectionExtract	151
7.5.3	ST_CollectionHomogenize	152
7.5.4	ST_CurveToLine	154
7.5.5	ST_Scroll	156
7.5.6	ST_FlipCoordinates	157
7.5.7	ST_Force2D	158
7.5.8	ST_Force3D	159
7.5.9	ST_Force3DZ	159
7.5.10	ST_Force3DM	160
7.5.11	ST_Force4D	161
7.5.12	ST_ForceCollection	162
7.5.13	ST_ForceCurve	163
7.5.14	ST_ForcePolygonCCW	164
7.5.15	ST_ForcePolygonCW	164
7.5.16	ST_ForceSFS	165
7.5.17	ST_ForceRHR	165
7.5.18	ST_LineExtend	166
7.5.19	ST_LineToCurve	167
7.5.20	ST_Multi	168
7.5.21	ST_Normalize	169
7.5.22	ST_Project	170
7.5.23	ST_QuantizeCoordinates	170
7.5.24	ST_RemovePoint	173
7.5.25	ST_RemoveRepeatedPoints	173
7.5.26	ST_RemoveIrrelevantPointsForView	174
7.5.27	ST_RemoveSmallParts	177
7.5.28	ST_Reverse	178
7.5.29	ST_Segmentize	179
7.5.30	ST_SetPoint	181
7.5.31	ST_ShiftLongitude	182
7.5.32	ST_WrapX	183
7.5.33	ST_SnapToGrid	184
7.5.34	ST_Snap	185
7.5.35	ST_SwapOrdinates	188
7.6	Geometry Validation	189

7.6.1	ST_IsValid	189
7.6.2	ST_IsValidDetail	190
7.6.3	ST_IsValidReason	192
7.6.4	ST_MakeValid	193
7.7	Spatial Reference System Functions	198
7.7.1	ST_InverseTransformPipeline	198
7.7.2	ST_SetSRID	199
7.7.3	ST_SRID	200
7.7.4	ST_Transform	201
7.7.5	ST_TransformPipeline	203
7.7.6	postgis_srs_codes	205
7.7.7	postgis_srs	206
7.7.8	postgis_srs_all	206
7.7.9	postgis_srs_search	207
7.8	Geometry Input	208
7.8.1	Well-Known Text (WKT)	208
7.8.1.1	ST_BdPolyFromText	208
7.8.1.2	ST_BdMPolyFromText	209
7.8.1.3	ST_GeogFromText	209
7.8.1.4	ST_GeographyFromText	210
7.8.1.5	ST_GeomCollFromText	210
7.8.1.6	ST_GeomFromEWKT	211
7.8.1.7	ST_GeomFromMARC21	213
7.8.1.8	ST_GeometryFromText	215
7.8.1.9	ST_GeomFromText	216
7.8.1.10	ST_LineFromText	217
7.8.1.11	ST_MLineFromText	218
7.8.1.12	ST_MPointFromText	219
7.8.1.13	ST_MPolyFromText	219
7.8.1.14	ST_PointFromText	220
7.8.1.15	ST_PolygonFromText	221
7.8.1.16	ST_WKTToSQL	222
7.8.2	Well-Known Binary (WKB)	223
7.8.2.1	ST_GeogFromWKB	223
7.8.2.2	ST_GeomFromEWKB	223
7.8.2.3	ST_GeomFromWKB	225
7.8.2.4	ST_LineFromWKB	226
7.8.2.5	ST_LinestringFromWKB	226
7.8.2.6	ST_PointFromWKB	227

7.8.2.7	ST_WKBToSQL	228
7.8.3	Other Formats	229
7.8.3.1	ST_Box2dFromGeoHash	229
7.8.3.2	ST_GeomFromGeoHash	230
7.8.3.3	ST_GeomFromGML	231
7.8.3.4	ST_GeomFromGeoJSON	233
7.8.3.5	ST_GeomFromKML	234
7.8.3.6	ST_GeomFromTWKB	235
7.8.3.7	ST_GMLToSQL	236
7.8.3.8	ST_LineFromEncodedPolyline	236
7.8.3.9	ST_PointFromGeoHash	237
7.8.3.10	ST_FromFlatGeobufToTable	238
7.8.3.11	ST_FromFlatGeobuf	238
7.9	Geometry Output	239
7.9.1	Well-Known Text (WKT)	239
7.9.1.1	ST_AsEWKT	239
7.9.1.2	ST_AsText	240
7.9.2	Well-Known Binary (WKB)	241
7.9.2.1	ST_AsBinary	241
7.9.2.2	ST_AsEWKB	243
7.9.2.3	ST_AsHEXEWKB	244
7.9.3	Other Formats	245
7.9.3.1	ST_AsEncodedPolyline	245
7.9.3.2	ST_AsFlatGeobuf	246
7.9.3.3	ST_AsGeobuf	246
7.9.3.4	ST_AsGeoJSON	247
7.9.3.5	ST_AsGML	249
7.9.3.6	ST_AsKML	253
7.9.3.7	ST_AsLatLonText	254
7.9.3.8	ST_AsMARC21	255
7.9.3.9	ST_AsMVTGeom	258
7.9.3.10	ST_AsMVT	259
7.9.3.11	ST_AsSVG	261
7.9.3.12	ST_AsTWKB	262
7.9.3.13	ST_AsX3D	263
7.9.3.14	ST_GeoHash	266
7.10	乱乱乱 (operator)	268
7.10.1	Bounding Box Operators	268
7.10.1.1	&&	268

7.10.1.2	&(geometry,box2df)	268
7.10.1.3	&(box2df,geometry)	269
7.10.1.4	&(box2df,box2df)	270
7.10.1.5	&&	271
7.10.1.6	&&(geometry,gidx)	272
7.10.1.7	&&(gidx,geometry)	273
7.10.1.8	&&(gidx,gidx)	274
7.10.1.9	&<	275
7.10.1.10	&<	276
7.10.1.11	&>	276
7.10.1.12	&<	277
7.10.1.13	&<	278
7.10.1.14		279
7.10.1.15	&>	280
7.10.1.16	@	281
7.10.1.17	@(geometry,box2df)	282
7.10.1.18	@(box2df,geometry)	283
7.10.1.19	@(box2df,box2df)	283
7.10.1.20	&>	284
7.10.1.21	>	285
7.10.1.22		286
7.10.1.23	(geometry,box2df)	287
7.10.1.24	(box2df,geometry)	287
7.10.1.25	(box2df,box2df)	288
7.10.1.26	=	289
7.10.2	表表表 (operator)	290
7.10.2.1	<->	290
7.10.2.2	=	292
7.10.2.3	<#>	293
7.10.2.4	<<->>	294
7.11	Spatial Relationships	295
7.11.1	Topological Relationships	295
7.11.1.1	ST_3DIntersects	295
7.11.1.2	ST_Contains	296
7.11.1.3	ST_ContainsProperly	300
7.11.1.4	ST_CoveredBy	301
7.11.1.5	ST_Covers	302
7.11.1.6	ST_Crosses	304
7.11.1.7	ST_Disjoint	306

7.11.1.8ST_Equals	307
7.11.1.9ST_Intersects	308
7.11.1.10ST_LineCrossingDirection	310
7.11.1.11ST_OrderingEquals	313
7.11.1.12ST_Overlaps	314
7.11.1.13ST_Relate	317
7.11.1.14ST_RelateMatch	319
7.11.1.15ST_Touches	320
7.11.1.16ST_Within	322
7.11.2Distance Relationships	324
7.11.2.1ST_3DDWithin	324
7.11.2.2ST_3DDFullyWithin	325
7.11.2.3ST_DFullyWithin	326
7.11.2.4ST_DWithin	327
7.11.2.5ST_PointInsideCircle	328
7.12Measurement Functions	329
7.12.1ST_Area	329
7.12.2ST_Azimuth	331
7.12.3ST_Angle	332
7.12.4ST_ClosestPoint	333
7.12.5ST_3DClosestPoint	335
7.12.6ST_Distance	336
7.12.7ST_3DDistance	338
7.12.8ST_DistanceSphere	339
7.12.9ST_DistanceSpheroid	340
7.12.10ST_FrechetDistance	341
7.12.11ST_HausdorffDistance	342
7.12.12ST_Length	344
7.12.13ST_Length2D	345
7.12.14ST_3DLength	346
7.12.15ST_LengthSpheroid	346
7.12.16ST_LongestLine	348
7.12.17ST_3DLongestLine	350
7.12.18ST_MaxDistance	351
7.12.19ST_3DMaxDistance	352
7.12.20ST_MinimumClearance	353
7.12.21ST_MinimumClearanceLine	354
7.12.22ST_Perimeter	354
7.12.23ST_Perimeter2D	356

7.12.2	ST_3DPerimeter	356
7.12.2	ST_ShortestLine	357
7.12.2	ST_3DShortestLine	359
7.13	Overlay Functions	360
7.13.1	ST_ClipByBox2D	360
7.13.2	ST_Difference	361
7.13.3	ST_Intersection	362
7.13.4	ST_MemUnion	365
7.13.5	ST_Node	365
7.13.6	ST_Split	366
7.13.7	ST_Subdivide	369
7.13.8	ST_SymDifference	372
7.13.9	ST_UnaryUnion	373
7.13.1	ST_Union	374
7.14	Geometric Functions	377
7.14.1	ST_Buffer	377
7.14.2	ST_BuildArea	382
7.14.3	ST_Centroid	383
7.14.4	ST_ChaikinSmoothing	385
7.14.5	ST_ConcaveHull	387
7.14.6	ST_ConvexHull	390
7.14.7	ST_DelaunayTriangles	392
7.14.8	ST_FilterByM	397
7.14.9	ST_GeneratePoints	398
7.14.1	ST_GeometricMedian	399
7.14.1	ST_LineMerge	401
7.14.1	ST_MaximumInscribedCircle	403
7.14.1	ST_LargestEmptyCircle	405
7.14.1	ST_MinimumBoundingCircle	407
7.14.1	ST_MinimumBoundingRadius	409
7.14.1	ST_OrientedEnvelope	409
7.14.1	ST_OffsetCurve	410
7.14.1	ST_PointOnSurface	414
7.14.1	ST_Polygonize	417
7.14.2	ST_ReducePrecision	419
7.14.2	ST_SharedPaths	420
7.14.2	ST_Simplify	422
7.14.2	ST_SimplifyPreserveTopology	424
7.14.2	ST_SimplifyPolygonHull	426

7.14.2ST_SimplifyVW	429
7.14.2ST_SetEffectiveArea	430
7.14.2ST_TriangulatePolygon	432
7.14.2ST_VoronoiLines	434
7.14.2ST_VoronoiPolygons	435
7.15Coverages	437
7.15.1ST_CoverageInvalidEdges	437
7.15.2ST_CoverageSimplify	438
7.15.3ST_CoverageUnion	440
7.15.4ST_CoverageClean	441
7.16Affine Transformations	442
7.16.1ST_Affine	442
7.16.2ST_Rotate	444
7.16.3ST_RotateX	445
7.16.4ST_RotateY	446
7.16.5ST_RotateZ	447
7.16.6ST_Scale	448
7.16.7ST_Translate	449
7.16.8ST_TransScale	451
7.17Clustering Functions	452
7.17.1ST_ClusterDBSCAN	452
7.17.2ST_ClusterIntersecting	454
7.17.3ST_ClusterIntersectingWin	454
7.17.4ST_ClusterKMeans	455
7.17.5ST_ClusterWithin	457
7.17.6ST_ClusterWithinWin	458
7.18Bounding Box Functions	459
7.18.1Box2D	459
7.18.2Box3D	460
7.18.3ST_EstimatedExtent	461
7.18.4ST_Expand	462
7.18.5ST_Extent	463
7.18.6ST_3DExtent	464
7.18.7ST_MakeBox2D	466
7.18.8ST_3DMakeBox	466
7.18.9ST_XMax	467
7.18.1ST_XMin	468
7.18.1ST_YMax	469
7.18.1ST_YMin	470

7.18.1	ST_ZMax	471
7.18.1	ST_ZMin	472
7.19	ST_LineInterpolatePoint (Linear Referencing)	473
7.19.1	ST_LineInterpolatePoint	473
7.19.2	ST_3DLineInterpolatePoint	474
7.19.3	ST_LineInterpolatePoints	475
7.19.4	ST_LineLocatePoint	476
7.19.5	ST_LineSubstring	477
7.19.6	ST_LocateAlong	479
7.19.7	ST_LocateBetween	480
7.19.8	ST_LocateBetweenElevations	482
7.19.9	ST_InterpolatePoint	483
7.19.10	ST_AddMeasure	483
7.20	Trajectory Functions	484
7.20.1	ST_IsValidTrajectory	484
7.20.2	ST_ClosestPointOfApproach	485
7.20.3	ST_DistanceCPA	486
7.20.4	ST_CPAWithin	487
7.21	Version Functions	488
7.21.1	PostGIS_Extensions_Upgrade	488
7.21.2	PostGIS_Full_Version	489
7.21.3	PostGIS_GEOS_Version	489
7.21.4	PostGIS_GEOS_Compiled_Version	490
7.21.5	PostGIS_Liblwgeom_Version	490
7.21.6	PostGIS_LibXML_Version	491
7.21.7	PostGIS_LibJSON_Version	491
7.21.8	PostGIS_Lib_Build_Date	492
7.21.9	PostGIS_Lib_Version	492
7.21.10	PostGIS_PROJ_Version	493
7.21.11	PostGIS_PROJ_Compiled_Version	493
7.21.12	PostGIS_Wagyu_Version	494
7.21.13	PostGIS_Scripts_Build_Date	495
7.21.14	PostGIS_Scripts_Installed	495
7.21.15	PostGIS_Scripts_Released	496
7.21.16	PostGIS_Version	496
7.22	PostGIS GUC(Grand Unified Custom Variable)	497
7.22.1	postgis.gdal_datapath	497
7.22.2	postgis.gdal_enabled_drivers	498
7.22.3	postgis.enable_outdb_rasters	499

7.22.4	postgis.gdal_vsi_options	500
7.22.5	postgis.gdal_cpl_debug	501
7.23	Troubleshooting Functions	501
7.23.1	PostGIS_AddBBox	501
7.23.2	PostGIS_DropBBox	502
7.23.3	PostGIS_HasBBox	502

8 SFCGAL Functions Reference 504

8.1	SFCGAL Management Functions	504
8.1.1	postgis_sfcgal_version	504
8.1.2	postgis_sfcgal_full_version	504
8.2	SFCGAL Accessors and Setters	505
8.2.1	CG_ForceLHR	505
8.2.2	CG_IsPlanar	505
8.2.3	CG_IsSolid	506
8.2.4	CG_MakeSolid	506
8.2.5	CG_Orientation	507
8.2.6	CG_Area	507
8.2.7	CG_3DArea	508
8.2.8	CG_Volume	508
8.2.9	ST_ForceLHR	509
8.2.10	ST_IsPlanar	510
8.2.11	ST_IsSolid	510
8.2.12	ST_MakeSolid	511
8.2.13	ST_Orientation	511
8.2.14	ST_3DArea	512
8.2.15	ST_Volume	513
8.3	SFCGAL Processing and Relationship Functions	514
8.3.1	CG_Intersection	514
8.3.2	CG_Intersects	515
8.3.3	CG_3DIntersects	515
8.3.4	CG_Difference	516
8.3.5	ST_3DDifference	517
8.3.6	CG_3DDifference	518
8.3.7	CG_Distance	519
8.3.8	CG_3DDistance	520
8.3.9	ST_3DConvexHull	521
8.3.10	CG_3DConvexHull	521
8.3.11	ST_3DIntersection	522

8.3.12CG_3DIntersection	523
8.3.13CG_Union	525
8.3.14ST_3DUnion	526
8.3.15CG_3DUnion	526
8.3.16ST_AlphaShape	528
8.3.17CG_AlphaShape	528
8.3.18CG_ApproxConvexPartition	531
8.3.19ST_ApproximateMedialAxis	532
8.3.20CG_ApproximateMedialAxis	533
8.3.21ST_ConstrainedDelaunayTriangles	534
8.3.22CG_ConstrainedDelaunayTriangles	535
8.3.23ST_Extrude	536
8.3.24CG_Extrude	536
8.3.25CG_ExtrudeStraightSkeleton	538
8.3.26CG_GreeneApproxConvexPartition	539
8.3.27ST_MinkowskiSum	540
8.3.28CG_MinkowskiSum	541
8.3.29ST_OptimalAlphaShape	543
8.3.30CG_OptimalAlphaShape	544
8.3.31CG_OptimalConvexPartition	546
8.3.32CG_StraightSkeleton	547
8.3.33ST_StraightSkeleton	549
8.3.34ST_Tessellate	550
8.3.35CG_Tessellate	551
8.3.36CG_Triangulate	553
8.3.37CG_Visibility	554
8.3.38CG_YMonotonePartition	555
8.3.39CG_StraightSkeletonPartition	556
8.3.40CG_3DBuffer	557
8.3.41CG_Rotate	559
8.3.42CG_2DRotate	560
8.3.43CG_3DRotate	560
8.3.44CG_RotateX	561
8.3.45CG_RotateY	562
8.3.46CG_RotateZ	562
8.3.47CG_Scale	563
8.3.48CG_3DScale	563
8.3.49CG_3DScaleAroundCenter	564
8.3.50CG_Translate	564
8.3.51CG_3DTranslate	565
8.3.52CG_Simplify	565
8.3.53CG_3DAlphaWrapping	568

9 国国 (topology)	572
9.1 国国国国	572
9.1.1 getfaceedges_returntype	572
9.1.2 TopoGeometry	573
9.1.3 validate_topology_returntype	573
9.2 国国国国国	574
9.2.1 TopoElement	574
9.2.2 TopoElementArray	574
9.3 国国国 TopoGeometry 国国	575
9.3.1 AddTopoGeometryColumn	575
9.3.2 RenameTopoGeometryColumn	576
9.3.3 DropTopology	577
9.3.4 RenameTopology	577
9.3.5 DropTopoGeometryColumn	578
9.3.6 Populate_Topology_Layer	578
9.3.7 TopologySummary	579
9.3.8 ValidateTopology	580
9.3.9 ValidateTopologyRelation	583
9.3.10 ValidateTopologyPrecision	583
9.3.11 MakeTopologyPrecise	584
9.3.12 FindTopology	584
9.3.13 FindLayer	585
9.3.14 TotalTopologySize	586
9.3.15 UpgradeTopology	586
9.4 Topology Statistics Management	587
9.5 国国国国国	587
9.5.1 CreateTopology	587
9.5.2 CopyTopology	588
9.5.3 ST_InitTopoGeo	589
9.5.4 ST_CreateTopoGeo	590
9.5.5 TopoGeo_AddPoint	591
9.5.6 TopoGeo_AddLineString	591
9.5.7 TopoGeo_AddPolygon	592
9.5.8 TopoGeo_LoadGeometry	592
9.6 国国国国国	593
9.6.1 ST_AddIsoNode	593
9.6.2 ST_AddIsoEdge	593
9.6.3 ST_AddEdgeNewFaces	594
9.6.4 ST_AddEdgeModFace	595

9.6.5	ST_RemEdgeNewFace	595
9.6.6	ST_RemEdgeModFace	596
9.6.7	ST_ChangeEdgeGeom	597
9.6.8	ST_ModEdgeSplit	598
9.6.9	ST_ModEdgeHeal	598
9.6.10	ST_NewEdgeHeal	599
9.6.11	ST_MoveIsoNode	599
9.6.12	ST_NewEdgesSplit	600
9.6.13	ST_RemoveIsoNode	601
9.6.14	ST_RemoveIsoEdge	602
9.7	Topology	602
9.7.1	GetEdgeByPoint	602
9.7.2	GetFaceByPoint	603
9.7.3	GetFaceContainingPoint	604
9.7.4	GetNodeByPoint	604
9.7.5	GetTopologyID	605
9.7.6	GetTopologySRID	606
9.7.7	GetTopologyName	606
9.7.8	ST_GetFaceEdges	607
9.7.9	ST_GetFaceGeometry	608
9.7.10	GetRingEdges	609
9.7.11	GetNodeEdges	609
9.8	TopologyUtils	610
9.8.1	Polygonize	610
9.8.2	AddNode	610
9.8.3	AddEdge	611
9.8.4	AddFace	612
9.8.5	ST_Simplify	614
9.8.6	RemoveUnusedPrimitives	615
9.9	TopoGeometry	615
9.9.1	CreateTopoGeom	615
9.9.2	toTopoGeom	617
9.9.3	TopoElementArray_Agg	618
9.9.4	TopoElement	619
9.10	TopoGeometryUtils	619
9.10.1	clearTopoGeom	619
9.10.2	TopoGeom_addElement	620
9.10.3	TopoGeom_remElement	620
9.10.4	TopoGeom_addTopoGeom	621

9.10.5toTopoGeom	622
9.11TopoGeometry 函数	622
9.11.1GetTopoGeomElementArray	622
9.11.2GetTopoGeomElements	622
9.11.3ST_SRID	623
9.12TopoGeometry 函数	624
9.12.1AsGML	624
9.12.2AsTopoJSON	626
9.13函数	627
9.13.1Equals	627
9.13.2Intersects	628
9.14Importing and exporting Topologies	629
9.14.1Using the Topology exporter	629
9.14.2Using the Topology importer	629
10函数, 函数	631
10.1函数	631
10.1.1raster2pgsql 函数	631
10.1.1.1Example Usage	631
10.1.1.2raster2pgsql options	632
10.1.2PostGIS 函数	633
10.1.3Using "out db" cloud rasters	634
10.2函数	635
10.2.1函数	635
10.2.2函数	636
10.3PostGIS 函数	637
10.3.1函数 ST_AsPNG 函数 PHP 函数	637
10.3.2函数 ST_AsPNG 函数 ASP.NET C# 函数	638
10.3.3函数 Java 函数	639
10.3.4PLPython 函数 SQL 函数	641
10.3.5PSQL 函数	641
11函数	643
11.1函数	644
11.1.1geomval	644
11.1.2addbandarg	644
11.1.3rastbandarg	644
11.1.4raster	645
11.1.5reclassarg	645

11.1.6summarystats	646
11.1.7unionarg	646
11.2表名表	647
11.2.1AddRasterConstraints	647
11.2.2DropRasterConstraints	649
11.2.3AddOverviewConstraints	650
11.2.4DropOverviewConstraints	651
11.2.5PostGIS_GDAL_Version	651
11.2.6PostGIS_Raster_Lib_Build_Date	652
11.2.7PostGIS_Raster_Lib_Version	652
11.2.8ST_GDALDrivers	653
11.2.9UpdateRasterSRID	658
11.2.10ST_CreateOverview	658
11.3表名表 (constructor)	659
11.3.1ST_AddBand	659
11.3.2ST_AsRaster	661
11.3.3ST_AsRasterAgg	663
11.3.4ST_Band	664
11.3.5ST_MakeEmptyCoverage	666
11.3.6ST_MakeEmptyRaster	667
11.3.7ST_Tile	668
11.3.8ST_Retile	671
11.3.9ST_FromGDALRaster	671
11.4表名表 (accessor)	672
11.4.1ST_GeoReference	672
11.4.2ST_Height	673
11.4.3ST_IsEmpty	674
11.4.4ST_MemSize	674
11.4.5ST_MetaData	675
11.4.6ST_NumBands	676
11.4.7ST_PixelHeight	676
11.4.8ST_PixelWidth	677
11.4.9ST_ScaleX	679
11.4.10ST_ScaleY	679
11.4.11ST_RasterToWorldCoord	680
11.4.12ST_RasterToWorldCoordX	681
11.4.13ST_RasterToWorldCoordY	682
11.4.14ST_Rotation	683
11.4.15ST_SkewX	683

11.4.16	ST_SkewY	684
11.4.17	ST_SRID	685
11.4.18	ST_Summary	685
11.4.19	ST_UpperLeftX	686
11.4.20	ST_UpperLeftY	687
11.4.21	ST_Width	687
11.4.22	ST_WorldToRasterCoord	688
11.4.23	ST_WorldToRasterCoordX	689
11.4.24	ST_WorldToRasterCoordY	689
11.5	ST_BandMetadata	690
11.5.1	ST_BandNoDataValue	692
11.5.2	ST_BandIsNoData	692
11.5.3	ST_BandPath	694
11.5.4	ST_BandFileSize	694
11.5.5	ST_BandFileTimestamp	695
11.5.6	ST_BandPixelType	695
11.5.7	ST_MinPossibleValue	696
11.5.8	ST_HasNoBand	697
11.6	ST_PixelAsPolygon (setter)	697
11.6.1	ST_PixelAsPolygons	698
11.6.2	ST_PixelAsPoint	699
11.6.3	ST_PixelAsPoints	700
11.6.4	ST_PixelAsCentroid	701
11.6.5	ST_PixelAsCentroids	701
11.6.6	ST_Value	703
11.6.7	ST_NearestValue	706
11.6.8	ST_SetZ	707
11.6.9	ST_SetM	709
11.6.10	ST_Neighborhood	710
11.6.11	ST_SetValue	712
11.6.12	ST_SetValues	713
11.6.13	ST_DumpValues	721
11.6.14	ST_PixelOfValue	722
11.7	ST_SetGeoReference	724
11.7.1	ST_SetRotation	725
11.7.2	ST_SetScale	726

11.7.4	ST_SetSkew	727
11.7.5	ST_SetSRID	728
11.7.6	ST_SetUpperLeft	728
11.7.7	ST_Resample	729
11.7.8	ST_Rescale	730
11.7.9	ST_Reskew	732
11.7.10	ST_SnapToGrid	733
11.7.11	ST_Resize	734
11.7.12	ST_Transform	735
11.8	乱乱乱乱乱乱乱乱	738
11.8.1	ST_SetBandNoDataValue	738
11.8.2	ST_SetBandIsNoData	739
11.8.3	ST_SetBandPath	741
11.8.4	ST_SetBandIndex	742
11.9	乱乱乱乱乱乱乱乱乱乱	744
11.9.1	ST_Count	744
11.9.2	ST_CountAgg	744
11.9.3	ST_Histogram	746
11.9.4	ST_Quantile	747
11.9.5	ST_SummaryStats	749
11.9.6	ST_SummaryStatsAgg	751
11.9.7	ST_ValueCount	753
11.10	Raster Inputs	755
11.10.1	ST_RastFromWKB	755
11.10.2	ST_RastFromHexWKB	756
11.11	乱乱乱乱乱	757
11.11.1	ST_AsBinary/ST_AsWKB	757
11.11.2	ST_AsHexWKB	757
11.11.3	ST_AsGDALRaster	758
11.11.4	ST_AsJPEG	759
11.11.5	ST_AsPNG	760
11.11.6	ST_AsTIFF	761
11.12	乱乱乱乱乱乱乱乱	762
11.12.1	ST_Clip	762
11.12.2	ST_ColorMap	766
11.12.3	ST_Grayscale	769
11.12.4	ST_Intersection	771
11.12.5	ST_MapAlgebra (callback function version)	773
11.12.6	ST_MapAlgebra (expression version)	779

11.12.\$T_MapAlgebraExpr	782
11.12.\$T_MapAlgebraExpr	784
11.12.\$T_MapAlgebraFct	789
11.12.\$T_MapAlgebraFct	793
11.12.\$T_MapAlgebraFctNgb	797
11.12.\$T_Reclass	799
11.12.\$T_ReclassExact	801
11.12.\$T_Union	803
11.13.乱乱乱乱乱乱乱乱乱乱	804
11.13.\$T_Distinct4ma	804
11.13.\$T_InvDistWeight4ma	805
11.13.\$T_Max4ma	806
11.13.\$T_Mean4ma	807
11.13.\$T_Min4ma	808
11.13.\$T_MinDist4ma	809
11.13.\$T_Range4ma	810
11.13.\$T_StdDev4ma	811
11.13.\$T_Sum4ma	812
11.14.乱乱乱乱乱乱乱乱乱乱	813
11.14.\$T_Aspect	813
11.14.\$T_HillShade	815
11.14.\$T_Roughness	817
11.14.\$T_Slope	818
11.14.\$T_TPI	819
11.14.\$T_TRI	820
11.14.\$T_InterpolateRaster	821
11.14.\$T_Contour	822
11.15.乱乱乱乱乱乱乱乱乱乱	823
11.15.Box3D	823
11.15.\$T_ConvexHull	823
11.15.\$T_DumpAsPolygons	824
11.15.\$T_Envelope	826
11.15.\$T_MinConvexHull	826
11.15.\$T_Polygon	827
11.15.\$T_IntersectionFractions	829
11.16.乱乱乱乱乱乱乱乱乱乱	830
11.16.&&	830
11.16.&<	831
11.16.&>	831

11.16.4	832
11.16.5	833
11.16.6	833
11.16.7	834
11.17	834
11.17.1	834
11.17.2	834
11.17.3	835
11.17.4	836
11.17.5	837
11.17.6	838
11.17.7	839
11.17.8	840
11.17.9	841
11.17.10	842
11.17.11	843
11.17.12	844
11.17.13	845
11.17.14	846
11.18	847
11.18.1	847
11.18.1.1	847
11.18.1.2	848
11.18.1.3	848
12	850
12.1	850
12.1.1	850
12.1.2	851
12.1.2.1	851
12.1.3	851
12.1.3.1	851
12.1.3.2	854
12.1.3.3	855
12.1.4	855
12.1.4.1	855
12.1.4.2	857
12.1.4.3	858
12.2	859

12.2.1Drop_Indexes_Generate_Script	860
12.2.2Drop_Nation_Tables_Generate_Script	861
12.2.3Drop_State_Tables_Generate_Script	861
12.2.4Geocode	862
12.2.5Geocode_Intersection	865
12.2.6Get_Geocode_Setting	866
12.2.7Get_Tract	867
12.2.8Install_Missing_Indexes	868
12.2.9Loader_Generate_Census_Script	868
12.2.10Loader_Generate_Script	870
12.2.11Loader_Generate_Nation_Script	872
12.2.12Missing_Indexes_Generate_Script	873
12.2.13Normalize_Address	874
12.2.14Pgcnorm Normalize_Address	875
12.2.15Print_Addy	877
12.2.16Reverse_Geocode	878
12.2.17Topology_Load_Tiger	880
12.2.18Set_Geocode_Setting	882

13 PostGIS Special Functions Index 884

13.1PostGIS Aggregate Functions	884
13.2PostGIS Window Functions	885
13.3PostGIS SQL-MM Compliant Functions	885
13.4PostGIS Geography Support Functions	889
13.5PostGIS Raster Support Functions	891
13.6PostGIS Geometry / Geography / Raster Dump Functions	897
13.7PostGIS Box Functions	897
13.8PostGIS Functions that support 3D	898
13.9PostGIS Curved Geometry Support Functions	904
13.10PostGIS Polyhedral Surface Support Functions	907
13.11PostGIS Function Support Matrix	911
13.12New, Enhanced or changed PostGIS Functions	930
13.12.1PostGIS Functions new or enhanced in 3.6	930
13.12.2PostGIS Functions new or enhanced in 3.5	931
13.12.3PostGIS Functions new or enhanced in 3.4	933
13.12.4PostGIS Functions new or enhanced in 3.3	934
13.12.5PostGIS Functions new or enhanced in 3.2	935
13.12.6PostGIS Functions new or enhanced in 3.1	937
13.12.7PostGIS Functions new or enhanced in 3.0	938

13.12. ~~B~~ ostGIS Functions new or enhanced in 2.5	940
13.12. ~~P~~ ostGIS Functions new or enhanced in 2.4	941
13.12. ~~P0~~ stGIS Functions new or enhanced in 2.3	943
13.12. ~~P1~~ stGIS Functions new or enhanced in 2.2	945
13.12. ~~P2~~ stGIS Functions new or enhanced in 2.1	948
13.12. ~~P3~~ stGIS Functions new or enhanced in 2.0	954
13.12. ~~P4~~ stGIS Functions new or enhanced in 1.5	965
13.12. ~~P5~~ stGIS Functions new or enhanced in 1.4	967
13.12. ~~P6~~ stGIS Functions new or enhanced in 1.3	967
14 Reporting Problems	968
14.1 Reporting Software Bugs	968
14.2 Reporting Documentation Issues	968
A Appendix	970
A.1 PostGIS 3.6.0	970
A.1.1 Breaking Changes	970
A.1.2 Removed / Deprecate signatures	970
A.1.3 New Features	971

Abstract

PostGIS 是 PostgreSQL 数据库的 GIS(地理信息系统) 扩展。PostGIS 使用 GiST 和 R-Tree 索引, GIS 数据存储在 PostgreSQL 数据库中。

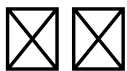


PostGIS 3.6.1dev 版本已发布。



PostGIS 3.6.1dev 版本已发布。PostGIS 3.0 版本已发布。PostGIS 3.0 版本已发布。
<https://postgis.net>

Chapter 1

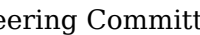
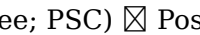
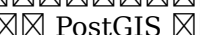


PostGIS is a spatial extension for the PostgreSQL relational database that was created by Refractions Research Inc, as a spatial database technology research project. Refractions is a GIS and database consulting company in Victoria, British Columbia, Canada, specializing in data integration and custom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

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

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

1.1

PostGIS  (Project Steering Committee; PSC)  PostGIS , , . PSC , PostGIS  , , PSC  API  PostGIS  .

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

  **(Regina Obe)** CI and website 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 ci maintenance.

  **(Paul Ramsey)** () 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, ci 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 贡献者 - 2020

Nicklas Avén 3D 支持, TWKB(Tiny WKB) 支持 (支持), 支持

Loïc Bartoletti SFCGAL enhancements and maintenance and ci 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, GeoBuf, and Flatgeobuf functions. Gitea testing and GitLab experimentation.

Aliaksandr Kalenik Geometry Processing, PostgreSQL gist, general bug fixing

1.3 贡献者 - 2019

Bhorie 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 支持, GDAL 支持, 支持

Olivier Courtin XML(KML, GML)/GeoJSON 支持, 3D 支持

Chris Hodgson PSC 成员. 支持, 支持, OSGeo 支持

Mateusz Loskot PostGIS CMake 支持, 支持, 支持 API 支持

Kevin Neufeld PSC 成员. 支持, 支持, PostGIS 支持, PostGIS 支持

Dave Blasby PostGIS 支持. 支持, 支持

Jeff Lounsbury shapefile 支持/支持. 支持 PostGIS 支持

Mark Leslie 支持. 支持, shapefile GUI 支持

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

David Zwarg 支持 (支持) 支持

1.4

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

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](#)
 - [HighGo](#)
 - [Hunter Systems Group](#)
 - [INIA-CSIC](#)
 - [ISciences, LLC](#)
 - [Kontur](#)
 - [Lidwala Consulting Engineers](#)
 - [LISAssoft](#)
 - [Logical Tracking & Tracing International AG](#)
 - [Maponics](#)
 - [Michigan Tech Research Institute](#)
 - [Natural Resources Canada](#)
 - [Norwegian Forest and Landscape Institute](#)
 - [Norwegian Institute of Bioeconomy Research \(NIBIO\)](#)
 - [OSGeo](#)
 - [Oslandia](#)
 - [Palantir Technologies](#)
 - [Paragon Corporation](#)
 - [Postgres Pro](#)
 - [R3 GIS](#)
 - [Refractions Research](#)
 - [Regione Toscana - SITA](#)
 - [Safe Software](#)
 - [Sirius Corporation plc](#)
 - [Stadt Uster](#)
 - [UC Davis Center for Vectorborne Diseases](#)
 - [Université Laval](#)
 - [U.S. Census Bureau](#)
 - [U.S. Department of State \(HIU\)](#)
 - [Zonar Systems](#)
-

1. 在开发过程中，我们使用了多种数据库系统，包括 MySQL、PostgreSQL、Oracle 和 Microsoft SQL Server。其中，PostGIS 是 PostgreSQL 的一个扩展，专门用于处理地理空间数据。

PostGIS 2.0.0 数据库引擎。 数据库引擎 PledgBank 数据库引擎。

```
postgistopology - 10 ██████████ 2.0.0 ██████ toTopGeometry ████████████████████
████ 250 ████████████████████.
```

```
postgis64windows - 20 64 PostGIS 100
PostgreSQL, PostGIS. PostgreSQL PostGIS 2.0.1 64
.
```

☐☐☐☐☐☐☐☐ 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

PostgreSQL DBMS - PostgreSQL DBMS
PostgreSQL DBMS, PostgreSQL DBMS, GiST DBMS, SQL DBMS

Chapter 2

PostGIS ☒ ☒

PostGIS

2.1 ☐ ☐ ☐ ☐

[illegible]

```
tar -xvzf postgres-3.6.1dev.tar.gz
cd postgres-3.6.1dev
./configure
make
make install
```

PostGIS (Section 3.3) (Section 3.4) .

2.2 ☒☒☒☒☒☒☒☒☒☒☒☒

Note

OS PostgreSQL/PostGIS . ,



Note!

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](#)

Stackbuilder [PostGIS Windows download site](#)
 1~2 very bleeding-edge windows experimental builds
 PostGIS

The PostGIS module is an extension to the PostgreSQL backend server. As such, PostGIS 3.6.1dev *requires* full PostgreSQL server headers access in order to compile. It can be built against PostgreSQL versions 12 - 18. Earlier versions of PostgreSQL are *not* supported.

Refer to the PostgreSQL installation guides if you haven't already installed PostgreSQL. <https://www.postgresql.org/docs/12/installation.html>

Note

GEOS 使用 PostgreSQL 使用 C++ 编译选项。



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

使用 C++ 编译选项。使用 PostgreSQL (使用 PostgreSQL 编译选项) 使用 PostgreSQL 编译选项。

PostGIS 使用 PostgreSQL 编译选项。使用 PostgreSQL 编译选项。

2.2.1 安装

PostGIS 使用 PostgreSQL 编译选项 <https://postgis.net/stuff/postgis-3.6.1dev.tar.gz> 编译选项。

```
wget https://postgis.net/stuff/postgis-3.6.1dev.tar.gz
tar -xvzf postgis-3.6.1dev.tar.gz
cd postgis-3.6.1dev
```

使用 PostgreSQL 编译选项 postgis-3.6.1dev (使用 PostgreSQL 编译选项)。

使用 PostgreSQL 编译选项, [svn](http://svn.osgeo.org/postgis/trunk/) 使用 <http://svn.osgeo.org/postgis/trunk/> 编译选项 (checkout) 编译选项。

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

使用 PostgreSQL 编译选项 postgis-3.6.1dev 编译选项。

```
./configure
```

2.2.2 编译选项

PostGIS 使用 PostgreSQL 编译选项。

编译选项

- PostgreSQL 12 - 18. A complete installation of PostgreSQL (including server headers) is required. PostgreSQL is available from <https://www.postgresql.org> 18 .
For a full PostgreSQL / PostGIS support matrix and PostGIS/GEOS support matrix refer to <https://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS>
- GNU C 编译选项 (gcc). PostGIS 使用 ANSI C 编译选项 gcc 编译选项。
- GNU Make(gmake 使用 make). 使用 GNU make 使用 make 编译选项. make -v 编译选项. 编译选项 make 使用 PostGIS Makefile 编译选项。
- 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 3.8.0 or greater, but GEOS 3.14+ 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_GeomFromGM and ST_GeomFromKML). LibXML2 is available for download from <https://gitlab.gnome.org/GNOME/libxml2/-/releases>.
- JSON-C 0.9. JSON-C is used by ST_GeomFromGeoJson and GeoJSON. JSON-C is available for download from <https://github.com/json-c/json-c/releases/>.
- GDAL, version 3+ is preferred. This is required for raster support. <https://gdal.org/download.html>.
- PostgreSQL 12.1 or higher. PostgreSQL 12.1 or higher is required for raster support. PostgreSQL 12.1 or higher is available for download from <http://trac.osgeo.org/postgis/ticket/635>.

Dependencies

- Section 2.1 describes the dependencies for building PostGIS.
- shapefile to shp2pgsql-gui uses GTK(GTK+2.0, 2.8+). <http://www.gtk.org/>.
- SFCGAL, 1.4.1 or higher is required and 2.1+ is needed to be able to use all functionality. SFCGAL can be used to provide additional 2D and 3D advanced analysis functions to PostGIS cf Chapter 8. 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://sfcgal.org>) <https://gitlab.com/sfcgal/SFCGAL/>.
- In order to build the Section 12.1 you will also need PCRE 1 or 2 <http://www.pcre.org> (which generally is already installed on nix systems). Section 12.1 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 Wagy 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-wagy` during the configure step.
- CUnit(CUnit). <http://cunit.sourceforge.net/>
- DocBook(xsltproc) DocBook <http://www.docbook.org/>
- DBLatex(dblatex) PDF <http://dblatex.sourceforge.net/>
- ImageMagick(convert) <http://www.imagemagick.org/>

2.2.3 Building

PostGIS is built using Makefile. The Makefile is located in the `Makefile` directory.

./configure

The `./configure` script is used to configure the build. It will check for the presence of the required libraries and headers. The `./configure` script will also generate the `Makefile` for the build.

The `./configure` script has several options. The `--help` option will display the help message. The `--help=short` option will display the short help message.

```
--prefix=PREFIX PostgreSQL XXXXXX SQL XXXXXXXXXXXXXXX. XXXXX PostgreSQL XXXX
XXXXXXXXXXXXXXXXXXXX.
```



PostgreSQL <http://trac.osgeo.org/postgis/ticket/635>

```
--with-gettext=no  PostGIS  gettext  ,  .  
  gettext  .  
  http://trac.osgeo.org/postgis/ticket/748  .  
  : gettext  .  gettext  .  
  GUI  .
```

--with-sfcgal=PATH `PATH` PostGIS sfcgal `PATH` sfcgal-config `PATH`.

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

Note

PostGIS `SVN` `PostGIS`, `PostGIS`.

**./autogen.sh**

`configure` `PostGIS`.

`tar` `PostGIS` `configure` `./autogen.sh`.

2.2.4

`Makefile` `PostGIS`.

make

`"PostGIS was built successfully. Ready to install."`.

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

make comments

PostGIS 2.0 `PostGIS`. `PostGIS` (cheat sheet) `html` `PostGIS`. `xsltproc` `doc` `topology_cheatsheet.html`, `tiger_geocoder_cheatsheet.html`, `raster_cheatsheet.html`, `postgis_cheatsheet.html` 4 `PostGIS`.

`html` `pdf` `PostGIS / PostgreSQL Study Guides`.

make cheatsheets

2.2.5 PostGIS Extensions

PostgreSQL 9.1 `PostGIS` extensions.

`function descriptions`. `docbook`, `PostGIS`:

make comments

`tar` `tar` `comments`.

PostgreSQL 9.1 `extensions`. `extensions`.

```
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](#)

extension files OS specific PostGIS binaries. PostGIS binaries are installed to the same location as the PostgreSQL binaries.

extension files are installed to the PostgreSQL share/extension directory. PostgreSQL extensions are installed to the PostgreSQL share/extension directory. PostGIS extensions are installed to the PostgreSQL share/extension directory.

- extension files are installed to the PostgreSQL share/extension directory. postgis.control, postgis_topology.control.
- extension files are installed to the PostgreSQL share/extension directory. extensions/postgis/sql/*.sql, extensions/postgis_topology/sql/*.sql

Once you do that, you should see postgis, postgis_topology as available extensions in PgAdmin -> extensions.

psql extension files are installed to the PostgreSQL share/extension directory.

```
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	3.6.1dev	3.6.1dev
address_standardizer_data_us	3.6.1dev	3.6.1dev
postgis	3.6.1dev	3.6.1dev
postgis_raster	3.6.1dev	3.6.1dev
postgis_sfcgal	3.6.1dev	
postgis_tiger_geocoder	3.6.1dev	3.6.1dev
postgis_topology	3.6.1dev	

(6 rows)

extension files are installed to the PostgreSQL share/extension directory, installed_version is the version of the extension installed. PostGIS extension files are installed to the PostgreSQL share/extension directory. PgAdmin III 1.14 extension files are installed to the PostgreSQL share/extension directory.

extension files are installed to the PostgreSQL share/extension directory. sql files are installed to the PostgreSQL share/extension directory. PostGIS extension files are installed to the PostgreSQL share/extension directory.

```
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_raster;
CREATE EXTENSION postgis_sfcgal;
CREATE EXTENSION fuzzystmatch; --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;
```

PSQL 简体中文, 简体中文.

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

List of installed extensions

```
-[ RECORD 1 ]-----
Name          | postgis
Version       | 3.6.1dev
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       | 3.6.1dev
Schema        | tiger
Description   | PostGIS tiger geocoder and reverse geocoder
-[ RECORD 4 ]-----
Name          | postgis_topology
Version       | 3.6.1dev
Schema        | topology
Description   | PostGIS topology spatial types and functions
```

Warning



spatial_ref_sys, layer, topology 简体中文. 简体中文 postgis 简体中文 postgis_topology extension 简体中文. 简体中文 PosGIS 2.0.1 简体中文 srid 简体中文. 简体中文 trac 简体中文. extension 简体中文 CREATE EXTENSION 简体中文. 简体中文 PostgreSQL extension 简体中文.

简体中文 3.6.1dev 简体中文, 简体中文: postgis_upgrade_22_minor.sql, raster_upgrade_22_minor.sql, topology_upgrade_22_minor.sql.

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

2.2.6 安装

PostGIS 简体中文, 简体中文.

make check

PostgreSQL 简体中文. 简体中文.

**Note**

PostgreSQL, GEOS, Proj4 `LD_LIBRARY_PATH` `LD_LIBRARY_PATH` `LD_LIBRARY_PATH`.

**Caution**

make check `PATH` `PGPORT`. PostgreSQL `--with-pgconfig` `PATH`.

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

.
```



```

===== 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.
=====

```

TIGER 繁體中文, 繁體 PostgreSQL 繁體 PostGIS 繁體 fuzzystmatch 繁體
 繁體. address_standardizer 繁體 PostGIS 繁體 address_standardizer 繁體.

```

cd extensions/postgis_tiger_geocoder
make install
make installcheck

```

繁體:

```

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

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

```

2.2.7 安裝

PostGIS 安裝。

make install

安裝 --prefix 安裝 PostgreSQL 安裝。

- 安裝 (loader) 安裝 [prefix]/bin 安裝。
- postgis.sql 安裝 SQL 安裝 [prefix]/share/contrib 安裝。
- PostGIS 安裝 [prefix]/lib 安裝。

Run `postgis_comments.sql`, `raster_comments.sql` and `make comments` to install the comments, and run `sql` to install the SQL.

make comments-install



Note

xsltproc is required to process `postgis_comments.sql`, `raster_comments.sql`, `topology_comments.sql` and `sql`.

2.3 Installing the address_standardizer extension

The `address_standardizer` extension is a PostgreSQL extension that provides a standard address standardization function. It is included in PostGIS 2.2 and later. For more information, see Section 12.1.

The `Normalize Address` function in PostGIS uses the TIGER geocoder (geocoder) to standardize addresses. For more information, see Section 2.4.2. The geocoder is a building block for the `address_standardizer` extension, and it is used to standardize addresses.

The `PCRE` library is required for the `address_standardizer` extension. The `PCRE` library is a regular expression engine. For more information, see <http://www.pcre.org> and Section 2.2.3. The `PCRE` library is included in the `address_standardizer` extension, and it is used to standardize addresses.

The `address_standardizer` extension is installed by running the `CREATE EXTENSION` command.

The `SQL` command to install the `address_standardizer` extension is:

```
CREATE EXTENSION address_standardizer;
```

The `rules`, `gaz`, and `lex` files are required for the `address_standardizer` extension.

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

The result of the query is:

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. PSQL, pgAdmin or any other SQL client. `PostGIS` `fuzzystmatch`.

```
CREATE EXTENSION postgis;
CREATE EXTENSION fuzzystmatch;
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;
```

`postgis_tiger_geocoder`:

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

```
tiger.loader_platform | tiger.loader_variables
```

3. SQL:

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

:

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

4. `tiger.loader_platform`. `sh` (convention) `debbie`.

```
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';
```

`debbie` `pg`, `unzip`, `shp2pgsql`, `PSQL` `declare_sect`.

`loader_platform` (common case).

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. 下載, 解壓縮到任何目錄 PC 安裝 gisdata 目錄。 下載 TIGER 數據。 解壓縮, 將 tiger.loader_variables 移到 staging_fold 目錄。

7. gisdata 目錄 staging_fold 目錄 temp 目錄。 將 TIGER 數據 temp 目錄。

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)
```

This will only have data if you marked zcta5 to be loaded

```
SELECT count(*) FROM tiger_data.zcta5_all;
```

```
count
-----
   33931
(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_**

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. 執行以下命令：

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

15. 執行以下命令以安裝 TIGER 缺少的索引 (統計) 表。

```
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.zcta5;
vacuum (analyze, verbose) tiger.zip_lookup_base;
vacuum (analyze, verbose) tiger.zip_state;
vacuum (analyze, verbose) tiger.zip_state_loc;
```

2.4.2 安裝 TIGER 缺少的索引

執行以下命令以安裝 **Normalize Address** 缺少的索引。執行以下命令以安裝 **address_standardizer** 缺少的索引，Section 2.3 中描述了這些索引。

`postgis_tiger_geocoder` 安裝了 **Normalize Address** 和 **Pagc Normalize Address** 缺少的索引。執行以下命令以安裝 TIGER 缺少的索引，包括 **rules table** (`tiger.pagc_rules`)，**gaz table** (`tiger.pagc_gaz`)，**lex table** (`tiger.pagc_lex`) 等。

2.4.3 Required tools for tiger data loading

執行以下命令以安裝 **tiger** 缺少的索引，**Drop_State_Tables_Generate_Script** 等。

執行以下命令：

- 執行以下命令以安裝 **unzip** 工具。
 Unix 系統上執行 `unzip` 工具。
 Windows 系統上，請前往 <http://www.7-zip.org/> 下載 7-zip。

- PostGIS shp2pgsql
- `wget` Unix/Linux `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.

`Loader_Generate_Nation_Script` 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.

`Install_Missing_Indexes`:

```
SELECT install_missing_indexes();
```

`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;
```

`Drop_Nation_Tables_Generate_Script`

```
SELECT drop_nation_tables_generate_script();
```

`drop SQL`.

`SELECT nation load`. `Loader_Generate_Nation_Script`

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

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

- [illegible]

```
SELECT version();
```

```
RPM ██████████ rpm ████████████████████: rpm -qa | grep postgresql
```

2. PostgreSQL.

```
SELECT postgis_full_version();
```

PostgreSQL, Proj4 GEOS

1. `pg_config.h` `PGSQL_VERSION`, `PGSQL_VERSION` and `PGSQL_VERSION`.

Chapter 3

PostGIS Administration

3.1 Performance Tuning

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

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

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

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

In addition to the Postgres settings, PostGIS has some custom settings which are listed in [Section 7.22](#).

3.1.1 Startup

These settings are configured in `postgresql.conf`:

`constraint_exclusion`

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

`shared_buffers`

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

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

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

3.1.2 Runtime

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

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

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

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

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

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

max_parallel_workers_per_gather

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

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

3.2 Configuring raster support

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

As of PostGIS 2.1.3, out-of-db rasters and all raster drivers are disabled by default. In order to re-enable these, you need to set the following environment variables `POSTGIS_GDAL_ENABLED_DRIVERS` and `POSTGIS_ENABLE_OUTDB_RASTERS` in the server environment. For PostGIS 2.2, you can use the more cross-platform approach of setting the corresponding Section [7.22](#).

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 ☒☒☒☒☒☒☒☒

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-3.5/

# 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 "3.6.1dev";  
ALTER EXTENSION postgis_topology UPDATE TO "3.6.1dev";
```

If you get an error notice something like:

```
No migration path defined for b'...' to 3.6.1dev
```

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 "3.6.1dev" 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 "3.6.1devnext";  
ALTER EXTENSION postgis_topology UPDATE TO "3.6.1devnext";
```

**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 b'...' 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-3.6.1dev/*_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 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. Unexpected errors, if any, will be printed to the standard error stream by psql. Keep a log of those.

```
postgis_restore "/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. So, after having run :

```
postgis_restore "/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 GIS (OGC) Standards

4.1.1 OGC Geometry

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

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

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

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

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

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

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

An important property of geometry values is their spatial **extent** or **bounding box**, which the OGC model calls **envelope**. This is the 2 or 3-dimensional box which encloses the coordinates of a geometry.

It is an efficient way to represent a geometry's extent in coordinate space and to check whether two geometries interact.

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

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

4.1.1.1 Point

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

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

4.1.1.2 LineString

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

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

4.1.1.3 LinearRing

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

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

4.1.1.4 Polygon

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

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

4.1.1.5 MultiPoint

A MultiPoint is a collection of Points.

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

4.1.1.6 MultiLineString

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

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

4.1.1.7 MultiPolygon

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

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

4.1.1.8 GeometryCollection

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

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

4.1.1.9 PolyhedralSurface

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

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

4.1.1.10 Triangle

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

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

4.1.1.11 TIN

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

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

4.1.2 SQL-MM Part 3

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



Note

SQL-MM ██████████. ████████ **1E-8** ████
████.

4.1.2.1 CircularString

[illegible]

CIRCULARSTRING(0 0, 1 1, 1 0)

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

4.1.2.2 CompoundCurve

[illegible]

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

4.1.2.3 CurvePolygon

[illegible]

PostGIS 1.4.

```
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

MULTICURVE [geometry](#), [text](#), [geometrycollection](#).

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

4.1.2.5 MultiSurface

MULTISURFACE [geometry](#), ([text](#)) [geometrycollection](#).

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

4.1.3 OpenGIS WKB [vs](#) WKT

OpenGIS [Specification](#): Well-Known Text (WKT) [and](#) Well-Known Binary (WKB) [formats](#). WKT [vs](#) WKB [comparison](#).

[WKT](#) (Well-Known Text) [formats](#). [WKT](#) SRS [formats](#):

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

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);
```

[OGC](#) [Specification](#):

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

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

- POINT(0 0)

WKB: 010100000000000000000000F03F000000000000F03

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

WKB: 010200000002000000000000000000004000000000000000400000000000002240000000000000

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);
```

OGC `POINT` WKB:

```
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".

OpenGIS `SRID` (SRID) `SRID`. `SRID` `SRID`.

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

4.2.1 OpenGIS WKB `WKT`

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

4.3.2 PostGIS 地理数据类型

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.

地理数据类型支持使用 SRID 来指定空间参考系统。地理数据类型支持使用 SRID 来指定空间参考系统。地理数据类型支持使用 SRID 来指定空间参考系统。

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

地理数据类型支持使用 SRID 来指定空间参考系统。地理数据类型支持使用 SRID 来指定空间参考系统。地理数据类型支持使用 SRID 来指定空间参考系统。

地理数据类型支持使用 SRID 来指定空间参考系统。地理数据类型支持使用 SRID 来指定空间参考系统。地理数据类型支持使用 SRID 来指定空间参考系统。

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

Great Circle mapper 地理数据类型支持使用 SRID 来指定空间参考系统。地理数据类型支持使用 SRID 来指定空间参考系统。地理数据类型支持使用 SRID 来指定空间参考系统。

```
-- 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 性能调优和最佳实践

本章提供了一些关于如何优化 PostGIS 性能的建议。这些建议旨在帮助您在不同的硬件配置和数据库配置下，获得最佳的性能表现。

本章内容分为两部分：第一部分介绍了一些通用的性能调优建议，第二部分则针对特定的数据库配置提供了具体的建议。

- 使用合适的索引类型。对于点数据，使用 GIST 索引；对于多边形数据，使用 GIST 索引或 R-Tree 索引。
- 使用合适的查询计划。对于复杂的查询，使用 EXPLAIN 命令来查看查询计划，并根据需要调整查询计划。
- 使用合适的数据库配置。对于不同的数据库配置，使用不同的配置参数来优化性能。

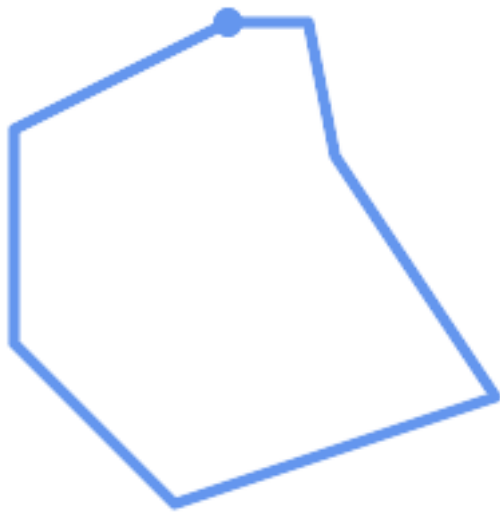
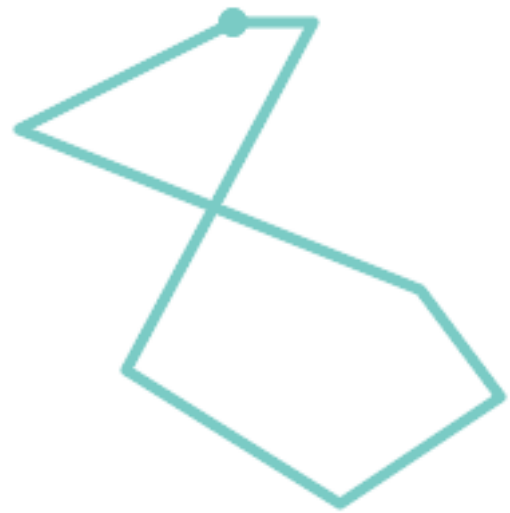
本章内容主要参考了 Section 13.11 和 Section 13.4 的相关内容。

4.3.4 常见问题 FAQ

1. 如何计算两个点之间的距离？
可以使用 `ST_Distance` 函数来计算两个点之间的距离。该函数返回的是两个点之间的欧几里得距离。
2. 如何计算一个多边形的面积？
可以使用 `ST_Area` 函数来计算一个多边形的面积。该函数返回的是多边形的面积。
3. 如何计算一个多边形的周长？
可以使用 `ST_Perimetre` 函数来计算一个多边形的周长。该函数返回的是多边形的周长。
4. 如何计算一个多边形的质心？
可以使用 `ST_Centroid` 函数来计算一个多边形的质心。该函数返回的是多边形的质心。

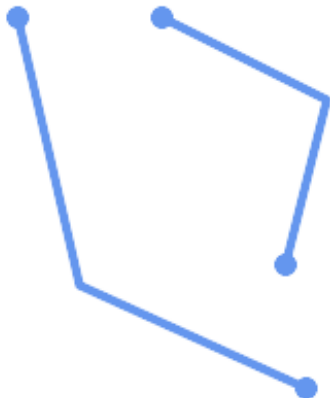
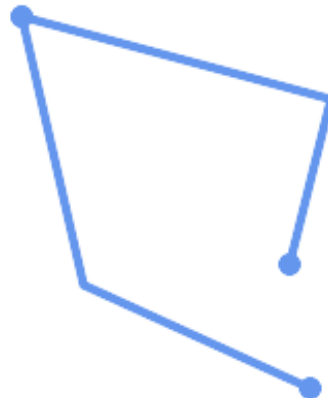
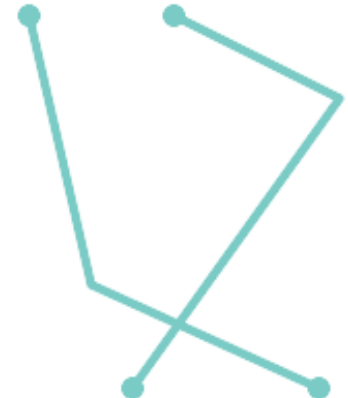
4.4 Geometry Validation

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

**(c)****(d)**

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

(e) and **(f)** are simple 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](#).

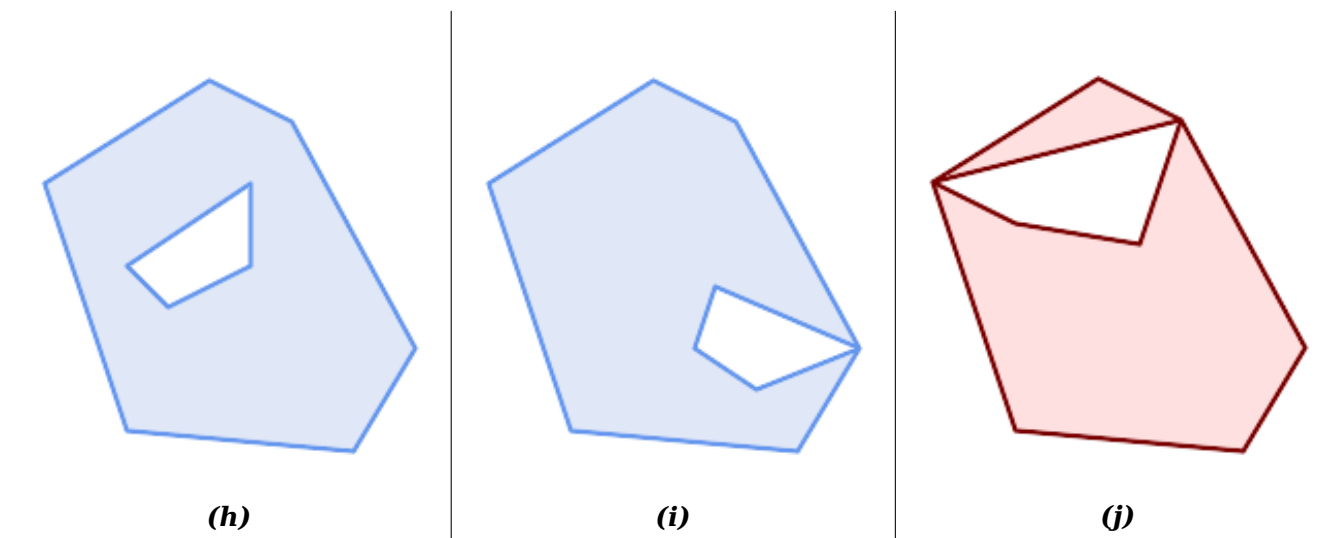
4.4.2 Valid Geometry

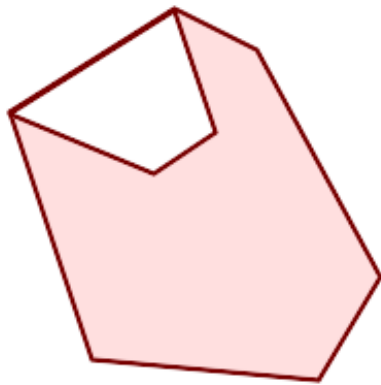
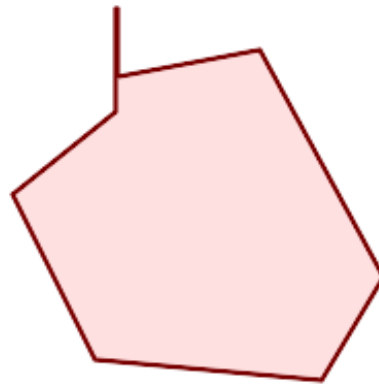
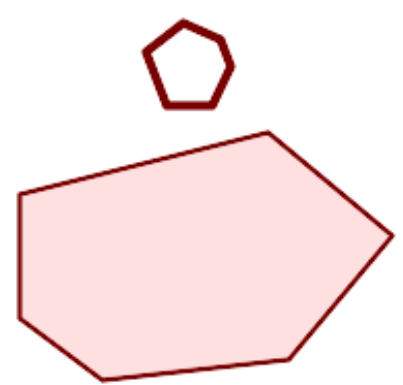
Geometry validity primarily applies to 2-dimensional geometries (POLYGONs and MULTIPOLYGONs). Validity is defined by rules that allow polygonal geometry to model planar areas unambiguously.

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.

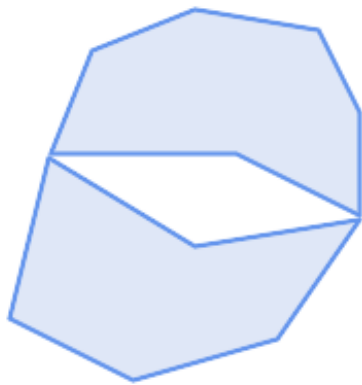
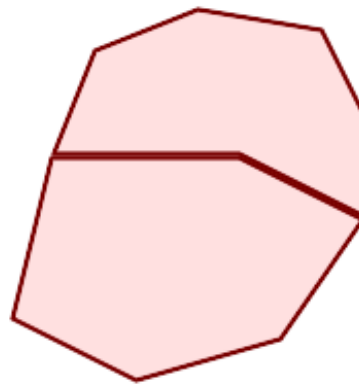
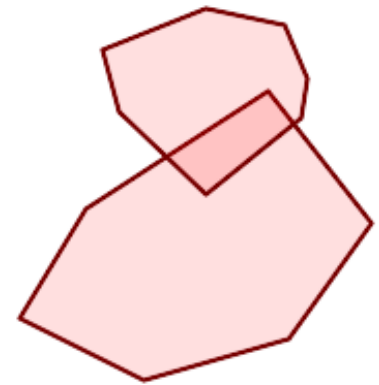


**(k)****(l)****(m)**

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.

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

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

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

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

len_nonzero	len_zero	self_int
-----+-----+-----		
t	f	t

POINT and MULTIPOINT geometries have no validity rules.

4.4.3 Managing Validity

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

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

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



Note

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

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

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

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

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

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

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

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

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

4.5 SPATIAL_REF_SYS

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_TransformPipe**. 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.

SPATIAL_REF_SYS:

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

Columns:

srid (SRS) identifier.

auth_name Authority name. If “EPSG” then 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 WKT(Well-Known Text) SRS.

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

```
ref sys.sql PROJ4TEXT SRTEXT PR0J4TEXT spatial_
```

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

地理参照系 (GCS) 是用于描述地理数据位置的坐标系。它由以下三个要素组成：

- 1. **SPATIAL_REF_SYS**：用于描述地理数据位置的坐标系。
- 2. **ESRI**：用于描述地理数据位置的坐标系。
- 3. spatialreference.org：用于描述地理数据位置的坐标系。

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 国国国国国国国国

4.6.1 国国国国国国国国

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**:

- 国国国国国国国国国国国国国国国国国国. POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON. 国国国国国国 Z, M 国 ZM 国国国国国国国国国国国国国国国国国国. 国国国, 国国国国国'LINESTRINGM' 国国国国国 3 国国国国国国国国国国国国国国国国国国, 国国国国国国国国国国国国国国国国国国. 国国国国国'POINTZM' 国国国国国国国国国国国国国国国国国国.
- 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 The GEOMETRY_COLUMNS VIEW

OpenGIS “SQL/MM ES: Simple Features Specification for SQL” GIS, <http://www.opengeospatial.org/standards/sfs>, <http://www.opengeospatial.org/standards/sfs/1.1.0/>. OpenGIS <http://www.opengeospatial.org/standards/sfs/1.1.0/>.

```
\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)	

Example:

f_table_catalog, f_table_schema, f_table_name The table catalog, schema, and table name. “f_table_catalog” is the catalog name. “f_table_schema” is the schema name. “f_table_name” is the table name. PostgreSQL <http://www.postgresql.org/docs/9.1/catalogs.html>. “f_table_name” is the table name. PostgreSQL <http://www.postgresql.org/docs/9.1/catalogs.html> (public schema).

f_geometry_column The geometry column name.

coord_dimension The dimension (2, 3, or 4) of the geometry.

srid The SRID (Spatial Reference ID) of the geometry. ID, SPATIAL_REF_SYS (foreign key).

type The geometry type. POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION, XYM, POINTM, LINESTRINGM, POLYGONM, MULTIPOINTM, MULTILINESTRINGM, MULTIPOLYGONM, GEOMETRYCOLLECTIONM. “GEOMETRY” is the geometry type.

4.6.3 geometry_columns

AddGeometryColumn() <http://www.postgresql.org/docs/9.1/functions-geometry.html>, SQL <http://www.postgresql.org/docs/9.1/functions-geometry.html> (bulk insert) <http://www.postgresql.org/docs/9.1/functions-geometry.html>. geometry_columns <http://www.postgresql.org/docs/9.1/functions-geometry.html>. PostGIS 2.0 <http://www.postgresql.org/docs/9.1/functions-geometry.html>, typmod <http://www.postgresql.org/docs/9.1/functions-geometry.html>.

```
-- 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_typedmod argument to false
SELECT populate_geometry_columns('myschema.my special pois'::regclass, false);
```

`\documentclass[12pt]{article}`, `\usepackage{typmod}`
`\geometry{columns=10cm}`. `\typmod{}`
`\documentclass[12pt]{article}`.

```
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);
```

PSQL ☒☒☒☒☒☒☒

```
\d pois ny;
```

[illegible]

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 (qid)
```

```

    pos = my prey
    Check constraints:

```

```
"enforce dims geom 2160" CHECK (st_ndims(geom 2160) = 2)
```

```

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)
```

geometry_columns 表。

```
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

-- 创建视图

```
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';
```

typmod 列, 列。

f_table_name	f_geometry_column	srid	type
vw_pois_ny_parks	geom	4326	POINT
vw_pois_ny_parks	geom_2160	0	GEOMETRY

PostGIS 表, 表。

```
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 GIS (表) 表

表, 表 GIS 表。表, 表 SQL 表 shapefile 表/表 PostgreSQL 表。

4.7.1 SQL 数据库操作

PostGIS 支持使用格式化的 SQL 语句。Oracle 风格的 SQL 语句，SQL 语句 "INSERT" 支持管道 (piping) 操作。

以下是一个示例 (roads.sql) 数据库操作：

```
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;
```

"psql" SQL 数据库操作 PostgreSQL 数据库。

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

4.7.2 shp2pgsql: ESRI shapefile 数据库操作

shp2pgsql 将 ESRI shapefile 文件，转换为 PostgreSQL 数据库。支持使用 SQL 语句。支持使用命令行 (command line) 操作。

shp2pgsql 支持使用 PostGIS 数据库。shp2pgsql-gui 支持使用 pgAdmin III 数据库。

c|a|d|p -- 数据库操作：

- c** 将 shapefile 文件转换为数据库。支持使用 SQL 语句。
- a** 将 shapefile 文件转换为数据库。支持使用 SQL 语句。
- d** 将 shapefile 文件 (drop) 转换为数据库。
- p** 将 SQL 语句转换为数据库。支持使用 SQL 语句。

-? 帮助信息。

-D 将 PostgreSQL " (dump)" 文件转换为数据库。支持使用 -a, -c 或 -d 选项。

-s [**<FROM_SRID>**:]**<SRID>** 将 shapefile 文件转换为数据库。支持使用 FROM_SRID 选项。支持使用 SRID 选项。

-k 将 shapefile 文件 (k, 数据库) 转换为数据库。支持使用 shapefile 文件。

4.8.2 使用pgsql2shp

pgsql2shp 是一个命令行工具，用于将 PostgreSQL 数据库中的表导出为 shapefile 文件。使用如下：

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

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

选项如下：

- f <filename> 输出文件的名称。
- h <host> 数据库主机名。
- p <port> 数据库端口号。
- P <password> 数据库密码。
- u <user> 数据库用户名。
- g <geometry column> 指定要导出的几何列名，shapefile 中几何列的名称。
- b 指定几何列的数据类型。默认为 GEOMETRY，也可以指定为 POINT、POLYGON、LINE 等。如果指定为非 GEOMETRY 类型，则需要使用 (cast) 指定数据类型。
- r 指定是否导出 raw 数据。gid 指定几何列的 ID 列名。
- m filename 指定要使用的地图文件（remap）文件。该文件用于指定如何将 PostgreSQL 中的几何数据类型映射到 shapefile 中的数据类型。VERYLONGSYMBOL SHORTONE ANOTHERVERYLONGSYMBOL SHORTER 等。

4.9 索引

索引是数据库中用于快速查找数据的一种方法。PostgreSQL 支持多种索引方法，包括 B-Tree、R-Tree、GiST 等。其中，B-Tree 索引是最常用的索引方法，适用于属性数据。R-Tree 索引适用于空间数据。GiST 索引适用于空间数据，并且支持范围查询。

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) 是一种用于空间数据的索引方法。它支持范围查询，并且可以用于存储和查询几何数据。PostGIS 使用 GiST 索引来存储和查询几何数据。R-Tree 也是一种用于空间数据的索引方法，但它不支持范围查询。
- **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 GiST

GiST is a “generalized search tree” index method, implemented in PostgreSQL. GIS indexes are implemented as B-Trees. The B-Trees store pointers to the actual geometry objects (or to other indexes) instead of the actual geometry objects.

GIS indexes are implemented as B-Trees. The B-Trees store pointers to the actual geometry objects (or to other indexes) instead of the actual geometry objects. This is efficient only if the data is physically ordered so that the bounding boxes for block ranges have minimal overlap (and ideally are contiguous).

“GiST” is a GiST index method implemented in PostgreSQL:

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

For 2D geometries, the `gist_geometry_ops_nd` operator class is used. In PostgreSQL 2.0, the `gist_geometry_ops` operator class was used, but it is deprecated.

```
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 it 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 GiST

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](#).

GIS indexes are implemented as B-Trees. The B-Trees store pointers to the actual geometry objects (or to other indexes) instead of the actual geometry objects. This is efficient only if the data is physically ordered so that the bounding boxes for block ranges have minimal overlap (and ideally are contiguous).

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 contiguous).

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;
```

” ” GiST :

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

2D . PostGIS 2.0 n , :

```
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

” ” GiST :

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

2D . PostGIS 2.0 n , :

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

PostgreSQL 12.1 introduced support for SP-GiST indexes. PostgreSQL 12.1 introduced support for SP-GiST indexes. PostgreSQL 12.1 introduced support for SP-GiST indexes. PostgreSQL 12.1 introduced support for SP-GiST indexes. PostgreSQL 12.1 introduced support for SP-GiST indexes.

PostgreSQL 12.1 introduced support for SP-GiST indexes. PostgreSQL 12.1 introduced support for SP-GiST indexes. PostgreSQL 12.1 introduced support for SP-GiST indexes. PostgreSQL 12.1 introduced support for SP-GiST indexes. PostgreSQL 12.1 introduced support for SP-GiST indexes.

- 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.

- `SET ENABLE_SEQSCAN=OFF` disables sequential scans. `SET ENABLE_SEQSCAN=OFF` disables sequential scans. `SET ENABLE_SEQSCAN=OFF` disables sequential scans. `SET ENABLE_SEQSCAN=OFF` disables sequential scans. `SET ENABLE_SEQSCAN=OFF` disables sequential scans.
- `random_page_cost` (cost) parameter, `postgresql.conf` `random_page_cost` parameter, `SET random_page_cost=#` parameter, `4` parameter, `1` parameter, `2` parameter. `random_page_cost` parameter, `4` parameter, `1` parameter, `2` parameter. `random_page_cost` parameter, `4` parameter, `1` parameter, `2` parameter.
- If **SET ENABLE_SEQSCAN TO OFF;** does not help your query, the query may be using a SQL construct that the Postgres planner is not yet able to optimize. It may be possible to rewrite the query in a way that the planner is able to handle. For example, a subquery with an inline SELECT may not produce an efficient plan, but could possibly be rewritten using a LATERAL JOIN.

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

Chapter 5

Spatial Queries

The *raison d'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 $\{0, 1, 2, F\}$:

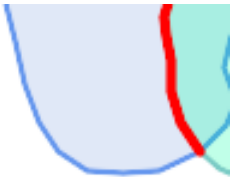
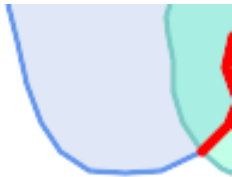
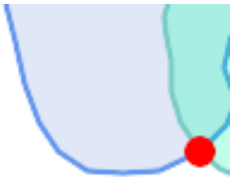
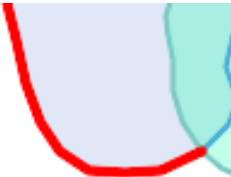
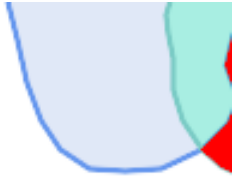
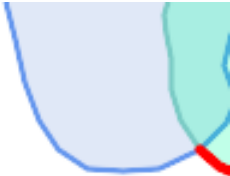
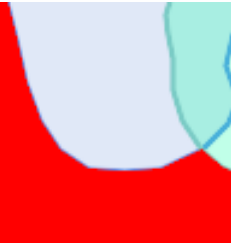
- 0 => point
- 1 => line
- 2 => area
- F => empty set

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	 $\dim(I(a) \cap I(b)) = 2$	 $\dim(I(a) \cap B(b)) = 1$	 $\dim(I(a) \cap E(b)) = 2$
	Boundary	 $\dim(B(a) \cap I(b)) = 1$	 $\dim(B(a) \cap B(b)) = 0$	 $\dim(B(a) \cap E(b)) = 1$
	Exterior	 $\dim(E(a) \cap I(b)) = 2$	 $\dim(E(a) \cap B(b)) = 1$	 $\dim(E(a) \cap E(b)) = 2$

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

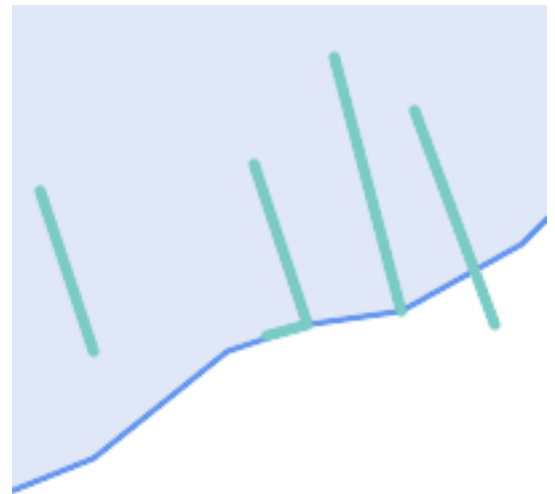
In some cases the named spatial relationships are insufficient to provide a desired spatial filter condition.



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

6.1.1 ☐ ☐ ☐ ☐ ☐

1. The TOAST algorithm is used to compress the data. The TOAST algorithm is a lossless data compression algorithm that uses a combination of run-length encoding and Huffman coding. The TOAST algorithm is used to compress the data into a single file. The TOAST algorithm is used to compress the data into a single file. The TOAST algorithm is used to compress the data into a single file.

~~~~~  
~~~~~ "EXPLAIN ANALYZE" PostgreSQL ~~~~~  
~~~~~ PostgreSQL ~~~~~:  
<http://archives.postgresql.org/pgsql-performance/2005-02/msg00030.php>

### 6.1.2 ☐ ☐ ☐ ☐

1. 在配置文件中添加以下配置项，并设置其值为 `on`。
 

```

        # 在配置文件中添加以下配置项，并设置其值为 on
        SET enable_seqscan TO on;
        
```

~~XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX~~. ~~XXXXXXXXXX~~" ~~XX~~" ~~XXXXXXXXXXXX~~  
~~XXXXXXXX~~, ~~XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX~~. ~~XXXXXXXXXXXXXXX~~, ~~XXXXXXXXXXXXXX~~ ~~XXXXXX~~:

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

```
geom_column bbox && [REDACTED]:
```

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

`mytable`, mytable `xxxxxx`, bbox " " .  
 (trigger) . `bbox` `xxxxxxxxxxx`,  
 UPDATE `xxxxxx`.

## 6.2

[illegible]

PostgreSQL 9.5.12 PostGIS 2.5.3 GiST 3.12.0. GiST 3.12.0  
 NULL 3.12.0. 3.12.0:

```
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.
```

HINT `isnull()`, `notnull()` "not null" `isnotnull()`:

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

`ALTER TABLE blubb ADD CHECK (geometry IS NOT NULL);`

### 6.3

3D 4D 2D OpenGL ST\_AsText() ST\_AsBinary() ST\_Force2D() ST\_Force3D() ST\_Force3DZ()

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

```
AddGeometryColumn() 向指定的表添加几何列。
语法: AddGeometryColumn(表名, 列名, 空间ID, 几何类型, 是否唯一, 是否主键)。
geometry_columns 表记录所有表的几何列信息。
```

CREATE TABLE test (
 geom GEOMETRY,
 name VARCHAR(255) WHERE geom IS NOT NULL
 )
 VACUUM;
 .
 \set test "WHERE dimension(the\_geom)>2"
 2D
 .



# Chapter 7

## PostGIS Reference

PostGIS is a spatial database, PostGIS is a spatial database.

**Note**

PostGIS is a spatial database, PostGIS is a spatial database. Spatial Type (ST) is a spatial database, PostGIS is a spatial database. ST\_ is a spatial database, PostGIS is a spatial database.

### 7.1 PostgreSQL PostGIS Geometry/Geography/Box

#### 7.1.1 box2d

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

box3d is a spatial database, postgis is a spatial database. ST\_3DExtent is a spatial database, box3d is a spatial database.

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).

|          |  |
|----------|--|
|          |  |
| box3d    |  |
| geometry |  |

Section 13.7

7.1.2 box3d

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

box3d is a geometry type in postgis. ST\_3DExtent 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).

|          |  |
|----------|--|
|          |  |
| box      |  |
| box2d    |  |
| geometry |  |

Section 13.7

7.1.3 geometry

geometry — geography

geography

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

|       |  |
|-------|--|
|       |  |
| box   |  |
| box2d |  |
| box3d |  |
| bytea |  |

|           |  |
|-----------|--|
| geography |  |
| text      |  |

Section 4.1, Section 4.3

7.1.4 geometry\_dump

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

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.

Section 13.6

7.1.5 geography

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

geography

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

|          |  |
|----------|--|
|          |  |
| geometry |  |



Section 4.3, Section 4.3

**7.2** ☐ ☐ ☐ ☐

### 7.2.1 AddGeometryColumn

AddGeometryColumn — .

## Synopsis

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

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

```
text AddGeometryColumn(varchar catalog_name, varchar schema_name, varchar table_name, var-
char column_name, integer srid, varchar type, integer dimension, boolean use typmod=true);
```



```

CREATE TABLE public.geometry_columns (
    ftable_name text NOT NULL,
    schema_name text NOT NULL,
    srid integer NOT NULL,
    spatial_ref_sys integer NOT NULL,
    type text NOT NULL,
    geometry_columns text NOT NULL,
    search_path text NOT NULL,
    SRID integer NOT NULL,
    spatial_ref_sys integer NOT NULL,
    geometry_columns text NOT NULL
);

```

### Note

```

2.0.0 geometry_columns geometry_columns ge-
ometry_columns . . . PostgreSQL . . . WGS84 POINT
ALTER TABLE some_table ADD COLUMN geom
geometry(Point,4326);

```

2.0.0. 使用 use\_typmod 选项, 使用

### Note

PostgreSQL: 2.0.0 安装。 geometry\_columns 数据类型, 使用 typmod 数据类型, 数据类型 geometry\_columns 数据类型, 使用 typmod 数据类型。 Section 4.6.3 数据类型。

- ✔ 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.

```

XXXXXXXX: 2.0.0 XX. use_tymod XXXXXXXXXXXX. XXXXXXXXXXXXXXXXXXXX tymod XXXX
XXXXXXXX.

```

❏❏

```
-- 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     | 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
```

| Column   | Type                 | Modifiers                                                               |
|----------|----------------------|-------------------------------------------------------------------------|
| id       | integer              | not null default nextval('my_schema.my_spatial_table_id_seq'::regclass) |
| geom     | geometry(Point,4326) |                                                                         |
| geom_c   | geometry             |                                                                         |
| geomcp_c | geometry             |                                                                         |

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

| Column   | Type                 | Modifiers                                                               |
|----------|----------------------|-------------------------------------------------------------------------|
| id       | integer              | not null default nextval('my_schema.my_spatial_table_id_seq'::regclass) |
| geom     | geometry(Point,4326) |                                                                         |
| geom_c   | geometry             |                                                                         |
| geomcp_c | geometry             |                                                                         |

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

-- geometry_columns view also registers the new columns --
SELECT f_geometry_column As col_name, type, srid, coord_dimension As ndims
FROM geometry_columns
WHERE f_table_name = 'my_spatial_table' AND f_table_schema = 'my_schema';
```

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



DropGeometryColumn, DropGeometryTable, Section 4.6.2, Section 4.6.3

### 7.2.2 DropGeometryColumn

DropGeometryColumn — □□□□□□□□□□□□□□□□□□□□.

## 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, var-
char column_name);
```



```

XXXXXXXXXXXXXXXXXXXXXXXXXXXX. schema_name X geometry_columns XXXXXXXXXXXXXXXXXXXX f_table_schema
XXXXXXXXXXXXXXXXXXXXXXXXXXXX.

```

- ✔ 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

```
mysql> ALTER TABLE geometry_columns ADD COLUMN srid INT(4) UNSIGNED ZEROFILL;  
mysql>
```



```
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;
```



### AddGeometryColumn, DropGeometryTable, Section 4.6.2

### 7.2.3 DropGeometryTable

DropGeometryTable — `geometry` columns `geometry`.

## 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);
```

ⓘ

`geometry_columns` ⚠️. `DropGeometryTable` (schema-aware) pgSQL `current_schema()` ⚠️.



### Note

⚠️: 2.0.0 ⚠️. `DropGeometryTable` ⚠️. `geometry_columns` ⚠️. `DropGeometryTable` ⚠️, `DropGeometryTable` ⚠️ `DROP TABLE` ⚠️.

ⓘ

```
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;
```

ⓘ

[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);
```

ⓘ

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.

ⓘ

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





```
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 meta-data.

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);

¶¶

¶¶¶¶¶¶¶, geometry\_columns ¶¶¶¶¶¶ srid ¶¶¶¶¶¶¶¶¶¶ SRID ¶¶¶¶¶¶. ¶¶: ¶¶¶¶¶¶¶¶¶¶¶¶ schema-aware postgres installations ¶¶¶¶¶¶ current\_schema() ¶¶



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



This method supports Circular Strings and Curves.

¶¶

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)
\.
```

¶¶¶¶¶¶¶¶¶¶ SRID ¶¶¶¶¶¶ SRID ¶ 4326 ¶¶¶¶¶¶¶¶¶¶:

```
SELECT UpdateGeometrySRID('roads','geom',4326);
```

¶¶¶¶¶¶¶ DDL ¶¶¶¶¶¶¶¶¶¶:

```
ALTER TABLE roads
  ALTER COLUMN geom TYPE geometry(MULTILINESTRING, 4326)
  USING ST_SetSRID(geom,4326);
```

¶¶¶¶¶¶¶¶¶¶ (¶¶'unknown' ¶¶) ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶, DDL ¶¶¶¶¶¶¶¶¶¶. ¶¶¶ PostGIS ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

```
ALTER TABLE roads
  ALTER COLUMN geom TYPE geometry(MULTILINESTRING, 3857) USING ST_Transform(ST_SetSRID(geom ↵
    ,4326),3857) ;
```

¶¶

**UpdateRasterSRID, ST\_SetSRID, ST\_Transform**

## 7.3 ¶¶¶¶¶¶ (constructor)

### 7.3.1 ST\_Collect

ST\_Collect — 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);
```

国国

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.

1.4.0 国国国国国国国国国国. 国国国国 ST\_MakeLine 国国国国国国国国国国. 国国国国国国国国国国 ST\_MakeLine 国国国国国国国国国国.



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



This method supports Circular Strings and Curves.

国国: **XLink** 国国

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))
```

□□ : □□□□□□□□

### 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))
```

☐ ☐ : ☐ ☐ ☐ ☐ ☐ ☐

### 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
```



ST Dump, ST AsBinary

### 7.3.2 ST\_LineFromMultiPoint

ST LineFromMultiPoint — □□□□□□□□□□□□□□□□□□□□.

## Synopsis

```
geometry ST_LineFromMultiPoint(geometry aMultiPoint);
```

[illegible]

Use **ST MakeLine** to create lines from Point or LineString inputs.



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

例

将MULTIPOINT(1 2 3, 4 5 6, 7 8 9)转换为LINESTRING。

```
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)
```

例

[ST\\_AsEWKT](#), [ST\\_AsKML](#)

### 7.3.3 ST\_MakeEnvelope

**ST\_MakeEnvelope** — 根据给定的边界框和SRID创建多边形。 指定的SRID 与SRS 相关联。

#### Synopsis

geometry **ST\_MakeEnvelope**(float xmin, float ymin, float xmax, float ymax, integer srid=unknown);

例

根据给定的边界框和SRID创建多边形。 指定的SRID 与SRS 相关联。 指定的SRID 与SRS 相关联。

1.5 版本之前。

2.0 版本之前 SRID 为 4326 (envelope) 的多边形。

例: 创建多边形

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

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

例

[ST\\_MakePoint](#), [ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_SRID](#)

### 7.3.4 ST\_MakeLine

**ST\_MakeLine** — 根据给定的点集创建线。

## Synopsis

```
geometry ST_MakeLine(geometry geom1, geometry geom2);
geometry ST_MakeLine(geometry[] geoms_array);
geometry ST_MakeLine(geometry set geoms);
```



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.

2.0.0 □□□□□□□□□□□□□□□□□□□□□□□□□□.

2.0.0 □□□□□□□□□□□□□□□□□□□□□□.

1.4.0 1.4.0. 1.4.0 ST\_MakeLine 1.4.0. 1.4.0  
1.4.0 ST\_MakeLine 1.4.0.

□□ : □□□□□□□□

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)

```

3D 2 BOX3D.

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

```

----- st_asewkt
LINESTRING(1 2 3,3 4 5)

```

[illegible]

```
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)
```











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) |








GPS 接收器，GPS 接收器 GPS 接收器

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;
```



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_MakePoint(float x, float y);
```

```
geometry ST_MakePoint(float x, float y, float z);
```

```
geometry ST_MakePoint(float x, float y, float z, float m);
```

☐☐

Creates a 2D XY, 3D XYZ or 4D XYZM Point geometry. Use **ST\_MakePointM** to make points with XYM coordinates.

Use **ST\_SetSRID** to specify a SRID for the created point.

While not OGC-compliant, ST\_MakePoint is faster 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



#### Note

The functions **ST\_Point**, **ST\_PointZ**, **ST\_PointM**, and **ST\_PointZM** can be used to create points with a given SRID.



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

☐☐

```
-- Create a point with unknown SRID
SELECT ST_MakePoint(-71.1043443253471, 42.3150676015829);

-- Create a point in the WGS 84 geodetic CRS
SELECT ST_SetSRID(ST_MakePoint(-71.1043443253471, 42.3150676015829),4326);

-- Create 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
```

☐☐

**ST\_GeomFromText**, **ST\_PointFromText**, **ST\_SetSRID**, **ST\_MakePointM**, **ST\_Point**, **ST\_PointZ**, **ST\_PointM**, **ST\_PointZM**

### 7.3.6 ST\_MakePointM

ST\_MakePointM — x, y 浮点型, 浮点型, 浮点型。

#### Synopsis

geometry **ST\_MakePointM**(float x, float y, float m);



国国

Creates a point with X, Y and M (measure) ordinates. Use **ST\_MakePoint** to make points with XY, XYZ, or XYZM coordinates.

Use **ST\_SetSRID** to specify a SRID for the created point.

**Note**

For geodetic coordinates, X is longitude and Y is latitude

**Note**

The functions **ST\_PointM**, and **ST\_PointZM** can be used to create points with an M value and a given SRID.

国国

**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)
```

x, y 国国国国国国国国国国国国国国国国.

```
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
```

国国

**ST\_MakePoint**, **ST\_SetSRID**, **ST\_PointM**, **ST\_PointZM**

### 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);
```

几何

几何 (shell) 几何. 几何.

**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

几何. 几何 **ST\_LineMerge** 几何 **ST\_Dump** 几何.



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

几何: 几何

几何.

```
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;
```

几何.

```
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))
```

例：刪除湖泊

刪除湖泊的 SQL 如下：

```
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

NULL 值會導致 [ST\\_MakePolygon](#) 函數返回 NULL 值。使用 CASE 語句可以避免此問題。

```
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;
```

例：

[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, float y);

geometry **ST\_Point**(float x, float y, integer srid=unknown);

国国

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

国国: 国国

```
SELECT ST_Point( -71.104, 42.315);
```

Creating a point with SRID specified:

```
SELECT ST_Point( -71.104, 42.315, 4326);
```

Alternative way of specifying SRID:

```
SELECT ST_SetSRID( ST_Point( -71.104, 42.315), 4326);
```

国国: 国国国

Create **geography** points using the :: cast syntax:

```
SELECT ST_Point( -71.104, 42.315, 4326)::geography;
```

Pre-PostGIS 3.2 code, using CAST:

```
SELECT CAST( ST_SetSRID(ST_Point( -71.104, 42.315), 4326) AS 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_Point( 3637510, 3014852, 2273), 4326)::geography;
```

国国

**ST\_MakePoint**, **ST\_PointZ**, **ST\_PointM**, **ST\_PointZM**, **ST\_SetSRID**, **ST\_Transform**

### 7.3.9 ST\_PointZ

ST\_PointZ — Creates a Point with X, Y, Z and SRID values.

#### Synopsis

geometry **ST\_PointZ**(float x, float y, float z, integer srid=unknown);

文档

ST\_PointZ 与 ST\_Point 类似。ST\_MakePoint 符合 OGC 规范。

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.

文档

```
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)
```

文档

[ST\\_MakePoint](#), [ST\\_PointFromText](#), [ST\\_SetSRID](#), [ST\\_MakePointM](#)

### 7.3.10 ST\_PointM

ST\_PointM — Creates a Point with X, Y, M and SRID values.

#### Synopsis

geometry **ST\_PointM**(float x, float y, float m, integer srid=unknown);

文档

ST\_PointM 与 ST\_Point 类似。ST\_MakePoint 符合 OGC 规范。

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.

文档

```
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)
```

[ST\\_MakePoint](#), [ST\\_PointFromText](#), [ST\\_SetSRID](#), [ST\\_MakePointM](#)

### 7.3.11 ST\_PointZM

**ST\_PointZM** — Creates a Point with X, Y, Z, M and SRID values.

#### Synopsis

geometry **ST\_PointZM**(float x, float y, float z, float m, integer srid=unknown);

[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.

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.

[ST\\_Point](#)

```
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)
```

[ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_PointM](#), [ST\\_PointZ](#), [ST\\_SetSRID](#)

[ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_PointM](#), [ST\\_PointZ](#), [ST\\_SetSRID](#)

### 7.3.12 ST\_Polygon

**ST\_Polygon** — Creates a Polygon from a LineString with a specified SRID.

#### Synopsis

geometry **ST\_Polygon**(geometry lineString, integer srid);

[ST\\_MakePolygon](#)

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**

ST\_LineMerge is not implemented in the `ST_Dump` function.



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



This method implements the SQL/MM specification. SQL-MM 3: 8.3.2



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

Examples

Create a 2D polygon.

```
SELECT ST_AsText( ST_Polygon('LINESTRING(75 29, 77 29, 77 29, 75 29)::geometry, 4326) );
-- result --
POLYGON((75 29, 77 29, 77 29, 75 29))
```

Create a 3D polygon.

```
SELECT ST_AsEWKT( ST_Polygon( ST_GeomFromEWKT('LINESTRING(75 29 1, 77 29 2, 77 29 3, 75 29 1)'), 4326) );
-- result --
SRID=4326;POLYGON((75 29 1, 77 29 2, 77 29 3, 75 29 1))
```

Related functions

[ST\\_AsEWKT](#), [ST\\_AsText](#), [ST\\_GeomFromEWKT](#), [ST\\_GeomFromText](#), [ST\\_LineMerge](#), [ST\\_MakePolygon](#)

### 7.3.13 ST\_TileEnvelope

**ST\_TileEnvelope** — Creates a rectangular Polygon in [Web Mercator](#) (SRID:3857) using the [XYZ tile system](#).

#### Synopsis

geometry **ST\_TileEnvelope**(integer tileZoom, integer tileX, integer tileY, geometry bounds=SRID=3857;LINESTRING(20037508.342789 -20037508.342789,20037508.342789 20037508.342789), float margin=0.0);

Examples

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](#).

国国国: 2.0.0 国国国国国国国国国 SRID 国国国国国国国国国.

2.1.0 国国国国国国国国国国.

国国: 国国国国国国国国国国

```
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))
```

国国

[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.

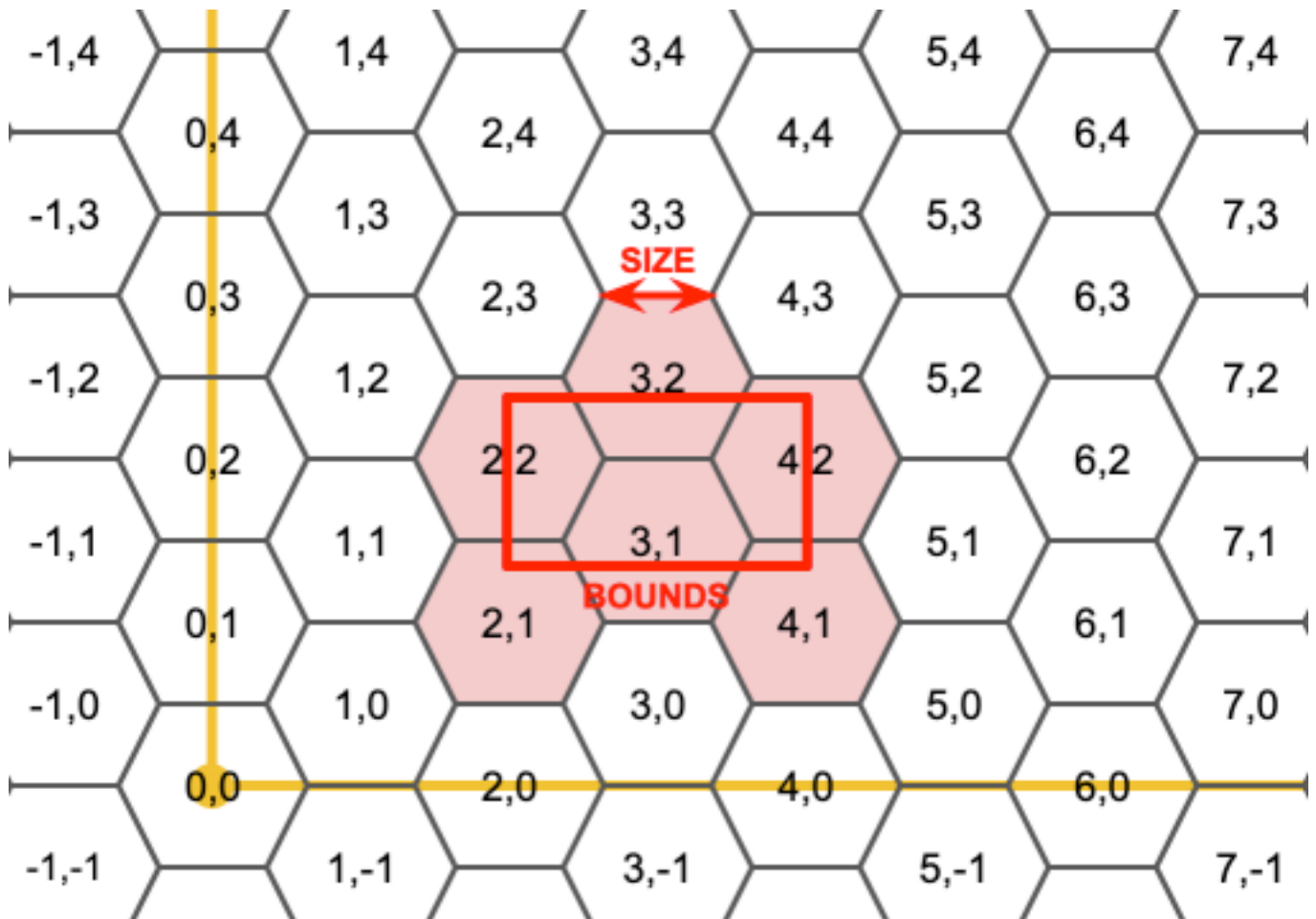
#### Synopsis

setof record **ST\_HexagonGrid**(float8 size, geometry bounds);

国国

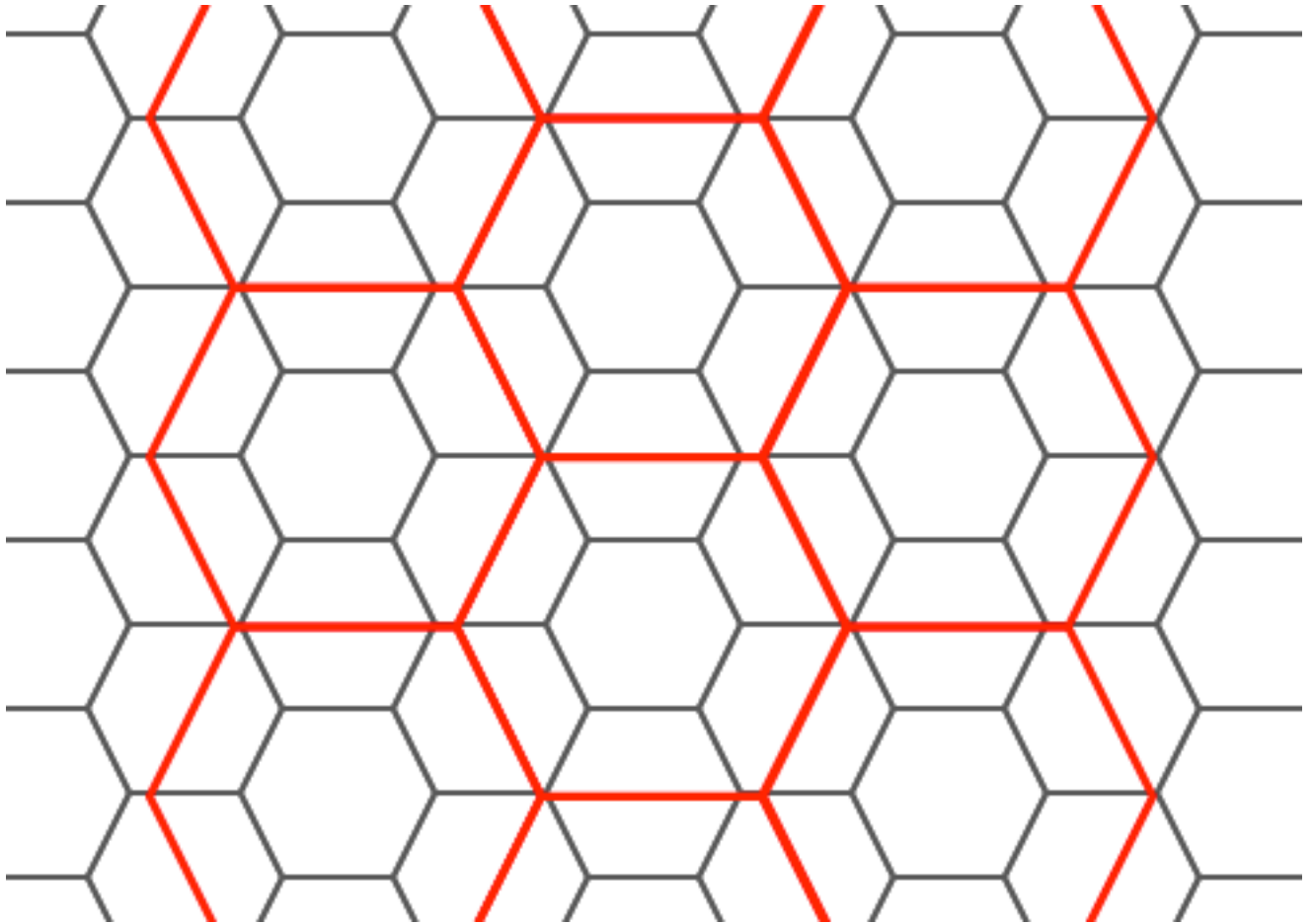
Starts with the concept of a hexagon tiling of the plane. (Not a hexagon tiling of the globe, this is not the [H3](#) tiling scheme.) For a given planar SRS, and a given edge size, starting at the origin of the SRS, there is one unique hexagonal tiling of the plane, `Tiling(SRS, Size)`. This function 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.



2.1.0 20240601.

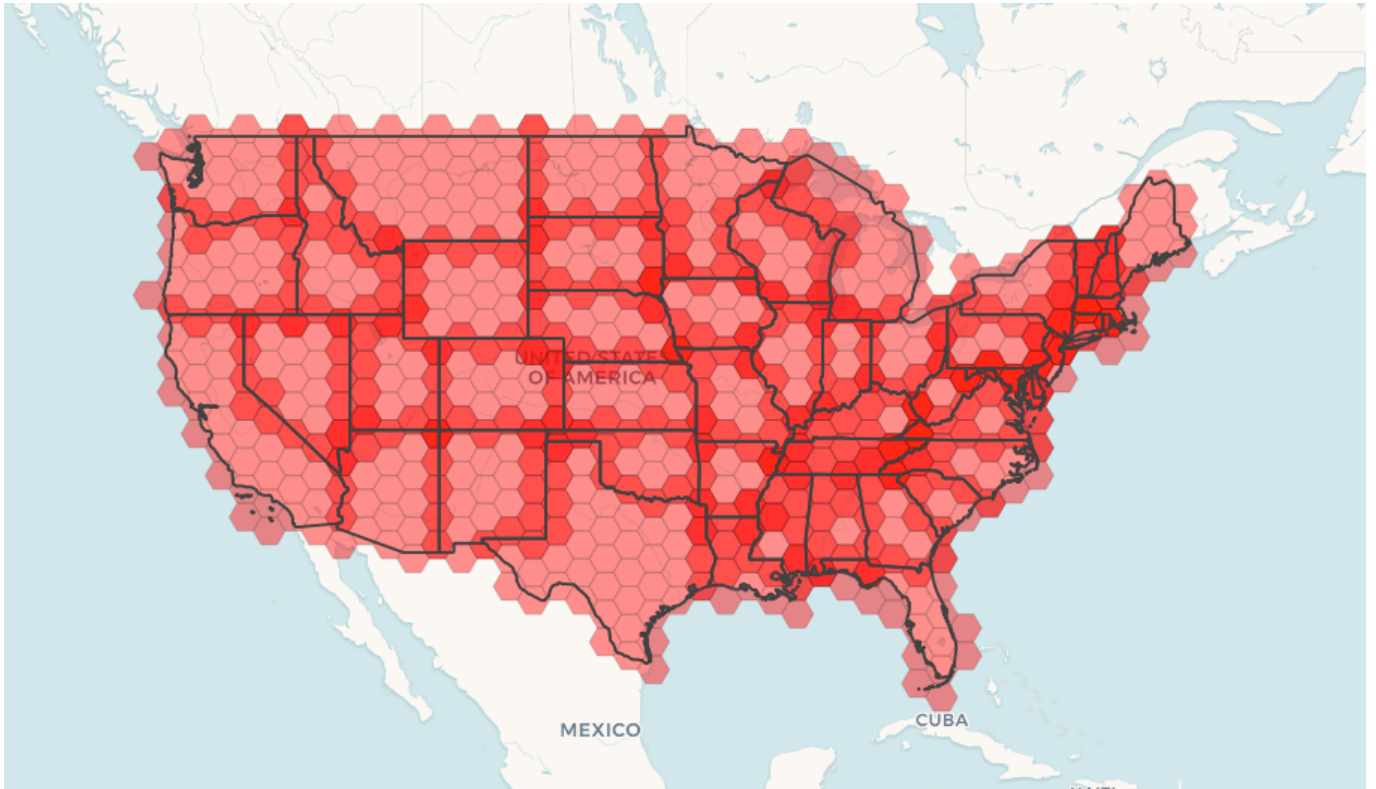
2.1: 20240601

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;
```

2.1: 20240601

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)
```



[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\_Hexagon**(float8 size, integer cell\_i, integer cell\_j, geometry origin);



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.

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))
```



[ST\\_TileEnvelope](#), [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

setof record **ST\_SquareGrid**(float8 size, geometry bounds);



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,  $\text{Tiling}(\text{SRS}, \text{Size})$ . This function answers the question: what grids in a given  $\text{Tiling}(\text{SRS}, \text{Size})$  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.

2.1.0 .

国国: 国国国国国国国国国国

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
```

国国: 国国国国国国国国

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
```

国国: 国国国国国国国国

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
```

国国

[ST\\_TileEnvelope](#), [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\_Square**(float8 size, integer cell\_i, integer cell\_j, geometry origin='POINT(0 0)');

☒☒

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 the SRID of the given origin. Use **ST\_SetSRID** to set the SRID if the given origin has an unknown SRID (as is the case by default).

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))
```

☒☒

**ST\_TileEnvelope**, **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);

☒☒

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.

Availability: 3.3.0

例: 生成字母 'Yo'

```
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);
```

例

[ST\\_AsTWKB](#), [ST\\_Scale](#), [ST\\_Translate](#)

## 7.4 几何类型 (accessor)

### 7.4.1 GeometryType

GeometryType — ST\_Geometry 几何类型的名称。

#### Synopsis

```
text GeometryType(geometry geomA);
```

例

ST\_GeometryType(ST\_GeomFromText('LINESTRING(0 0,1 1)', 4326)) --> 'LINESTRING'.

OGC 简单特征访问器 (SFA) 规范, 第 2.1.1.1 节, 描述了该函数的行为。



### Note

POINTM' .

Version: 2.0.0, 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).



```
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 g;
result
-----
TIN
```



文档

**ST\_GeometryType**

## 7.4.2 ST\_Boundary

ST\_Boundary — 返回几何对象的边界。

### Synopsis

geometry **ST\_Boundary**(geometry geomA);

文档

ST\_Boundary (closure) 返回几何对象的边界。 ST\_Boundary (combinatorial boundary) 返回 OGC 3.12.3.2 定义的边界。 ST\_Boundary (位相的) 返回 OGC 3.12.2 定义的边界 (primitive) 返回几何对象的边界。

GEOS 支持



### Note

2.0.0 版本中，GEOMETRYCOLLECTION 类型的边界返回 NULL。

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

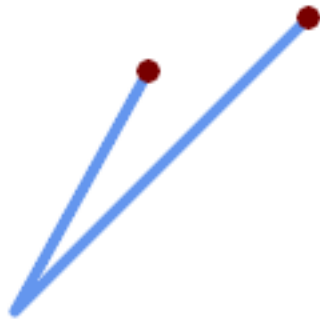
✓ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.17

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

文档: 2.1.0 版本中，边界返回 NULL。

Changed: 3.2.0 support for TIN, does not use geos, does not linearize curves

文档

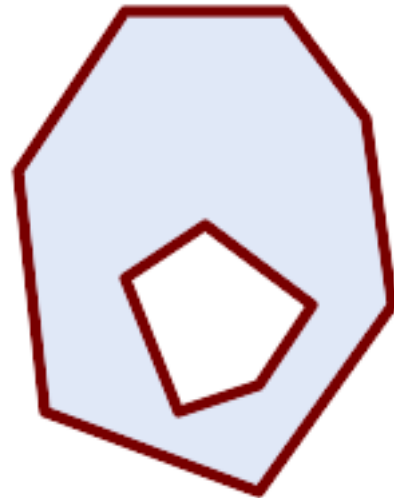


国国国国国国国国国国国国国国国国

```
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))
```



国国国国国国国国国国国国国国国国

```
SELECT ST_Boundary(geom)
FROM (SELECT
'POLYGON (( 10 130, 50 190, 110 190, 140
150, 150 80, 100 10, 20 40, 10 130 ),
( 70 40, 100 50, 120 80, 80 110,
50 90, 70 40 ))::geometry As geom) As f;
```

ST\_AsText output

```
MULTILINESTRING((10 130,50 190,110
190,140 150,150 80,100 10,20 40,10 130),
(70 40,100 50,120 80,80 110,50
90,70 40))
```

```
SELECT ST_AsText(ST_Boundary(ST_GeomFromText('LINESTRING(1 1,0 0, -1 1)')));
st_astext
-----
MULTIPOINT((1 1),(-1 1))

SELECT ST_AsText(ST_Boundary(ST_GeomFromText('POLYGON((1 1,0 0, -1 1, 1 1)'))));
st_astext
-----
LINESTRING(1 1,0 0,-1 1,1 1)

--Using a 3d polygon
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromEWKT('POLYGON((1 1 1,0 0 1, -1 1 1, 1 1 1)'))));
st_asewkt
-----
LINESTRING(1 1 1,0 0 1,-1 1 1,1 1 1)

--Using a 3d multilinestring
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromEWKT('MULTILINESTRING((1 1 1,0 0 0.5, -1 1 1),(1 1
0.5,0 0 0.5, -1 1 0.5, 1 1 0.5) )'))));
st_asewkt
-----
```

```
MULTIPOINT((-1 1 1),(1 1 0.75))
```

☐☐

[ST\\_AsText](#), [ST\\_ExteriorRing](#), [ST\\_MakePolygon](#)

### 7.4.3 ST\_BoundingDiagonal

`ST_BoundingDiagonal` — 返回几何图形的边界对角线。

#### Synopsis

geometry **ST\_BoundingDiagonal**(geometry geom, boolean fits=false);

☐☐

返回几何图形的边界对角线。如果几何图形是点，则返回该点。如果几何图形是线，则返回该线的两个端点。如果几何图形是面，则返回该面的两个最远端点。如果几何图形是集合，则返回该集合中所有几何图形的边界对角线的并集。

`fits` 参数（best fit）用于控制是否返回最佳拟合的对角线。如果为 true，则返回最佳拟合的对角线；如果为 false，则返回几何图形的实际边界对角线。

返回值的 SRID 与输入几何图形的 SRID 相同。



#### Note

对于非凸多边形，返回的对角线可能不是唯一的。在这种情况下，函数将返回最短的对角线。

2.2.0 版本引入。



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



This function supports M coordinates.

☐☐

```
-- 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
```

☐☐

[ST\\_StartPoint](#), [ST\\_EndPoint](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_Z](#), [ST\\_M](#), &&&

### 7.4.4 ST\_CoordDim

ST\_CoordDim — ST\_Geometry

#### Synopsis

integer **ST\_CoordDim**(geometry geomA);

ST\_Geometry

MM, **ST\_NDims**

- ✓ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.3
- ✓ This method supports Circular Strings and Curves.
- ✓ This function supports 3d and will not drop the z-index.
- ✓ This function supports Polyhedral surfaces.
- ✓ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

```
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
```

**ST\_NDims**

### 7.4.5 ST\_Dimension

ST\_Dimension — ST\_Geometry

#### Synopsis

integer **ST\_Dimension**(geometry g);

¶¶

¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. OGC ¶¶¶ s2.1.1.1 ¶¶¶¶¶ POINT ¶ 0, LINESTRING ¶ 1, POLYGON ¶ 2, ¶¶¶ GEOMETRYCOLLECTION ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. ¶¶¶¶¶¶ (¶¶) ¶¶¶¶¶ null ¶¶¶¶¶.



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

¶¶¶¶: 2.0.0 ¶¶¶¶¶¶¶¶¶ (polyhedral surface) ¶ TIN ¶¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.



#### Note

2.0.0 ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

¶¶

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

¶¶

**ST\_NDims**

## 7.4.6 ST\_Dump

**ST\_Dump** — Returns a set of `geometry_dump` rows for the components of a geometry.

### Synopsis

```
geometry_dump[] ST_Dump(geometry g1);
```

¶¶

A set-returning function (SRF) that extracts the components of a geometry. It returns a set of **geom-etry\_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\_Collect** / GROUP BY, in that it creates new rows. For example it can be use to expand MULTIPOLYGONS into POLYGONS.

¶¶¶¶: 2.0.0 ¶¶¶¶¶¶¶¶¶, ¶¶¶¶ TIN ¶¶¶¶¶¶¶¶¶¶¶¶.

Availability: PostGIS 1.0.0RC1. Requires PostgreSQL 7.3 or higher.



**Note**

1.3.4 (curve) 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.

```
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)
```

**, TIN**

```
-- Polyhedral surface example
-- Break a Polyhedral surface into its faces
SELECT (a.p_geom).path[1] As path, ST_AsEWKT((a.p_geom).geom) As geom_ewkt
FROM (SELECT ST_Dump(ST_GeomFromEWKT('POLYHEDRALSURFACE(
((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1
1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)') ) AS p_geom ) AS a;

path | geom_ewkt
-----+-----
1 | POLYGON((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0))
2 | POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))
3 | POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0))
4 | POLYGON((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0))
5 | POLYGON((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0))
6 | POLYGON((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))

-- TIN --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
```

```
(SELECT
  ST_Dump( ST_GeomFromEWKT('TIN (((
    0 0 0,
    0 0 1,
    0 1 0,
    0 0 0
  ))), ((
    0 0 0,
    0 1 0,
    1 1 0,
    0 0 0
  ))
)' ) AS gdump
) AS g;

-- result --
path | wkt
-----+-----
{1}  | TRIANGLE((0 0 0,0 0 1,0 1 0,0 0 0))
{2}  | TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))
```



geometry dump, ST\_GeomFromEWKT, ST\_Dump, ST\_GeometryN, ST\_NumGeometries

### 7.4.7 ST\_DumpPoints

ST DumpPoints — .

## Synopsis

```
geometry_dump[] ST_DumpPoints(geometry geom);
```



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.

Version: 2.0.0, TIN

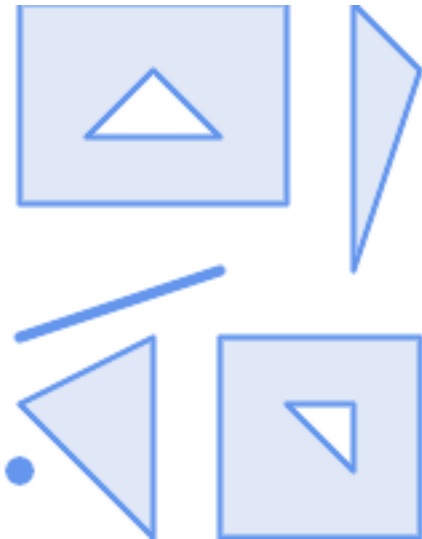
1.5.0 ☒☒☒☒☒☒☒☒☒☒☒.

- ✔ 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.

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)
```



```
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)

POLYHEDRAL SURFACE, TIN

```
-- 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)
```

`geometry_dump`, `ST_GeomFromEWKT`, `ST_Dump`, `ST_GeometryN`, `ST_NumGeometries`

7.4.8 ST\_DumpSegments

`ST_DumpSegments` —

Synopsis

```
geometry_dump[] ST_DumpSegments(geometry geom);
```

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).

- the *geom* field LINESTRINGS represent the linear segments of the supplied geometry, while the CIRCULARSTRINGS represent the arc segments.
- the *path* field (an integer[]) is an index enumerating the segment start point positions in the elements of the supplied geometry. The indices are 1-based. For example, for a LINESTRING the paths are {i} where i is the nth segment start point in the LINESTRING. For a POLYGON the paths are {i, j} where i is the ring number (1 is outer; inner rings follow) and j is the segment start point position in the ring.

Availability: 3.2.0

-  This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
-  This function supports 3d and will not drop the z-index.

```
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    | b'' b'' | st_astext           |
|---------|---------|---------------------|
| -----   |         |                     |
| {1,1}   | b'' b'' | LINESTRING(1 1,3 3) |
| {1,2}   | b'' b'' | LINESTRING(3 3,4 4) |
| {2,1,1} | b'' b'' | LINESTRING(5 5,6 6) |
| {2,1,2} | b'' b'' | LINESTRING(6 6,7 7) |
| {2,1,3} | b'' b'' | LINESTRING(7 7,5 5) |

(5 rows)

**TIN**

```
-- 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  | b'' b'' | st_astext           |
|-------|---------|---------------------|
| ----- |         |                     |
| {1,1} | b'' b'' | LINESTRING(0 0,0 9) |
| {1,2} | b'' b'' | LINESTRING(0 9,9 0) |
| {1,3} | b'' b'' | LINESTRING(9 0,0 0) |

(3 rows)

```
-- TIN --
SELECT path, ST_AsEWKT(geom)
FROM (
  SELECT (ST_DumpSegments(g.geom)).*
  FROM (SELECT 'TIN(((
    0 0 0,
    0 0 1,
    0 1 0,
    0 0 0
  )), ((
    0 0 0,
    0 1 0,
    1 1 0,
    0 0 0
  )))
  )'::geometry AS geom
        ) AS g
```

```
) j;

path  b''|b''      st_asewkt
-----
{1,1,1} b''|b''  LINESTRING(0 0 0,0 0 1)
{1,1,2} b''|b''  LINESTRING(0 0 1,0 1 0)
{1,1,3} b''|b''  LINESTRING(0 1 0,0 0 0)
{2,1,1} b''|b''  LINESTRING(0 0 0,0 1 0)
{2,1,2} b''|b''  LINESTRING(0 1 0,1 1 0)
{2,1,3} b''|b''  LINESTRING(1 1 0,0 0 0)
(6 rows)
```

[geometry\\_dump](#), [ST\\_Collect](#), [ST\\_Dump](#), [ST\\_NumInteriorRing](#),

### 7.4.9 ST\_DumpRings

**ST\_DumpRings** — Returns a set of [geometry\\_dump](#) rows for the exterior and interior rings of a Polygon.

#### Synopsis

`geometry_dump[] ST_DumpRings(geometry a_polygon);`

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**

ST\_DumpRings returns a set of geometry\_dump rows. ST\_Dump returns a set of geometry\_dump rows.

Availability: PostGIS 1.1.3. Requires PostgreSQL 7.3 or higher.



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

General form of query.

```
SELECT polyTable.field1, polyTable.field1,
       (ST_DumpRings(polyTable.geom)).geom As geom
FROM polyTable;
```

A polygon with a single hole.



国国

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)
```

ST\_LineString 国国 ST\_CircularString 国国国国国国国国国国国国国国国国.

```
SELECT ST_AsText(ST_EndPoint('CIRCULARSTRING(5 2,-3 1.999999, -2 1, -4 2, 6 3)::geometry')) ←
;
st_astext
-----
POINT(6 3)
```

国国

**ST\_PointN, ST\_StartPoint**

### 7.4.11 ST\_Envelope

ST\_Envelope — 国国国国国国国国国国 (double precision; float8) 国国国国国国国国国国国国国国国国.

#### Synopsis

geometry **ST\_Envelope**(geometry g1);

国国

国国国国国国国 float8 国国国国国国国国国国国国国国国国. 国国国国国国国国国国国国国国国国国国国国国国 ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)). (PostGIS 国 ZMIN/ZMAX 国国国国国国国国国国.)

国国国国 (国国国国, 国国国) 国国 POLYGON 国国国国国国, 国国国国 POINT 国国 LINESTRING 国国国国国国 国国国国.

1.5.0 国国国国国国国国国国, float4 国国国国国国国国国国国国国国国国国国国国国国国国国国.



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

国国

```

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))

```

国国

**Box2D, Box3D, ST\_OrientedEnvelope**

## 7.4.12 ST\_ExteriorRing

ST\_ExteriorRing — 国国国国国国国国国国国国国国国国.

### Synopsis

geometry **ST\_ExteriorRing**(geometry a\_polygon);

国国

POLYGON 国国国国国国 (exterior ring) 国国国国国国国国国国国国国国国国. 国国国国国国国国国国 NULL 国国国国国国.



### Note

国国国国国国国国国国国国国国国国. 国国国国国国 ST\_Dump 国国国国国国国国国国国国.

- ✔ 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.

国国

```

--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)
```



ST\_InteriorRingN, ST\_Boundary, ST\_NumInteriorRings

### 7.4.13 ST\_GeometryN

ST GeometryN — ST Geometry 

## Synopsis

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



00000000, (00) 000, (00) 000000, 0000 (multicurve) 00 (00) 0000000000  
 000 1-00 N 0000000000, 000000 NULL 000000.



Note!

### Note

0.8.0  OGC  1-OGC.  0-OGC



Note!

### Note

ST Dump , .

Version: 2.0.0, TIN

OpenGL: 2.0.0 NVIDIA 440.33 NULL GLSL 2.0.0 ST\_GeometryN(..,1) GLSL 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: 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).

## SQL

```
--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);
```

## SQL, TIN

```
-- 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
    ))
  )
```



```
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
```

☒☒

## GeometryType

### 7.4.15 ST\_HasArc

ST\_HasArc — Tests if a geometry contains a circular arc

#### Synopsis

boolean **ST\_HasArc**(geometry geomA);

☒☒

☒☒☒☒☒☒☒☒☒, ☒☒☒, ☒☒☒☒☒☒☒ TRUE ☒☒☒☒☒☒.

1.2.2 ☒☒☒☒☒☒☒☒☒☒☒☒.



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



This method supports Circular Strings and Curves.

国国

```
SELECT ST_HasArc(ST_Collect('LINESTRING(1 2, 3 4, 5 6)', 'CIRCULARSTRING(1 1, 2 3, 4 5, 6 6, 5 6)'));
      st_hasarc
      -----
      t
```

国国

[ST\\_CurveToLine](#), [ST\\_PointN](#)

### 7.4.16 ST\_InteriorRingN

`ST_InteriorRingN` — 国国国国国国国国国国国国国国国国.

#### Synopsis

geometry **ST\_InteriorRingN**(geometry a\_polygon, integer n);

国国

国国国国国 N 国国国国国国国国国国国国国国国国. 国国国国国国国国国国国 N 国国 (range) 国国国 NULL 国国国国国.



#### Note

国国国国国国国国国国国国国国国国. 国国国国国国 ST\_Dump 国国国国国国国国国国国.



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.

国国

```
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;
```

国国

[ST\\_ExteriorRing](#), [ST\\_M](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 7.4.17 ST\_NumCurves

**ST\_NumCurves** — Return the number of component curves in a `CompoundCurve`.

#### Synopsis

integer **ST\_NumCurves**(geometry a\_compoundcurve);



Return the number of component curves in a `CompoundCurve`, zero for an empty `CompoundCurve`, or NULL for a non-`CompoundCurve` input.



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.



```
-- Returns 3
SELECT ST_NumCurves('COMPOUNDCURVE(
  (2 2, 2.5 2.5),
  CIRCULARSTRING(2.5 2.5, 4.5 2.5, 3.5 3.5),
  (3.5 3.5, 2.5 4.5, 3 5, 2 2)
)');

-- Returns 0
SELECT ST_NumCurves('COMPOUNDCURVE EMPTY');
```



[ST\\_CurveN](#), [ST\\_Dump](#), [ST\\_ExteriorRing](#), [ST\\_NumInteriorRings](#), [ST\\_NumGeometries](#)

### 7.4.18 ST\_CurveN

**ST\_CurveN** — Returns the Nth component curve geometry of a `CompoundCurve`.

#### Synopsis

geometry **ST\_CurveN**(geometry a\_compoundcurve, integer index);



Returns the Nth component curve geometry of a `CompoundCurve`. The index starts at 1. Returns NULL if the geometry is not a `CompoundCurve` or the index is out of range.



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.

例

```
SELECT ST_AsText(ST_CurveN('COMPOUNDCURVE(
  (2 2, 2.5 2.5),
  CIRCULARSTRING(2.5 2.5, 4.5 2.5, 3.5 3.5),
  (3.5 3.5, 2.5 4.5, 3 5, 2 2)
)', 1));
```

例

[ST\\_NumCurves](#), [ST\\_Dump](#), [ST\\_ExteriorRing](#), [ST\\_NumInteriorRings](#), [ST\\_NumGeometries](#)

### 7.4.19 ST\_IsClosed

**ST\_IsClosed** — **LINESTRING** 是否是闭合的 **TRUE** 或 **FALSE**。多边形 (面) 总是 **TRUE**。

#### Synopsis

boolean **ST\_IsClosed**(geometry g);

例

**LINESTRING** 是否是闭合的 **TRUE**。多边形, 面 (面) 总是 **TRUE**。

- ✓ 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 defines the result of **ST\_IsClosed**(NULL) to be 0, while PostGIS returns NULL.

- ✓ This function supports 3d and will not drop the z-index.
- ✓ This method supports Circular Strings and Curves.
- 例: 2.0.0 多面体 (polyhedral surface)。
- ✓ This function supports Polyhedral surfaces.

例



```

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)

```

测试测试测试测试

```

-- 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

```

国国

**ST\_IsRing**

## 7.4.20 ST\_IsCollection

ST\_IsCollection — 国国国国国国国国国国, 国国国, 国国国国国国 TRUE 国国国国国国.

### Synopsis

boolean **ST\_IsCollection**(geometry g);

国国

国国国国国国国国国国国国国国 TRUE 国国国国国国:

- GEOMETRYCOLLECTION
- MULTI{POINT,POLYGON,LINestring,CURVE,SURFACE}
- COMPOUNDCURVE

**Note**

国国国国国国国国国国国国国国. 国国国国国国国国国国国国国国国国国国 TRUE 国国国国国国国国国国国国.



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



This method supports Circular Strings and Curves.

国国

```

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)

```



❏❏

```
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)
```

### 7.4.22 ST\_IsPolygonCCW

**ST\_IsPolygonCCW** — Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.

#### Synopsis

boolean **ST\_IsPolygonCCW** ( geometry geom );

❏❏

Returns true if all polygonal components of the input geometry use a counter-clockwise orientation for their exterior ring, and a clockwise direction for all interior rings.

Returns true if the geometry has no polygonal components.



#### Note

Closed linestrings are not considered polygonal components, so you would still get a true return by passing a single closed linestring no matter its orientation.

**Note**

If a polygonal geometry does not use reversed orientation for interior rings (i.e., if one or more interior rings are oriented in the same direction as an exterior ring) then both `ST_IsPolygonCW` and `ST_IsPolygonCCW` will return false.

2.2.0 `ST_IsPolygonCW`.



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



This function supports M coordinates.

`ST_IsPolygonCW`

`ST_ForcePolygonCW` , `ST_ForcePolygonCCW` , `ST_IsPolygonCW`

### 7.4.23 `ST_IsPolygonCW`

`ST_IsPolygonCW` — Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.

#### Synopsis

boolean **`ST_IsPolygonCW`** ( geometry geom );

`ST_IsPolygonCW`

Returns true if all polygonal components of the input geometry use a clockwise orientation for their exterior ring, and a counter-clockwise direction for all interior rings.

Returns true if the geometry has no polygonal components.

**Note**

Closed linestrings are not considered polygonal components, so you would still get a true return by passing a single closed linestring no matter its orientation.

**Note**

If a polygonal geometry does not use reversed orientation for interior rings (i.e., if one or more interior rings are oriented in the same direction as an exterior ring) then both `ST_IsPolygonCW` and `ST_IsPolygonCCW` will return false.

2.2.0 `ST_IsPolygonCW`.



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



This function supports M coordinates.



ST\_ForcePolygonCW , ST\_ForcePolygonCCW , ST\_IsPolygonCW

#### 7.4.24 ST\_IsRing

ST\_IsRing — Tests if a LineString is closed and simple.

## Synopsis

```
boolean ST_IsRing(geometry g);
```



Returns TRUE if this LINESTRING is both **ST\_IsClosed** (ST\_StartPoint(g) ~= ST\_Endpoint(g)) and **ST\_IsSimple** (does not self intersect).

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

✔ This method implements the SQL/MM specification. SQL-MM 3: 7.1.6



### Note

SQL-MM defines the result of ST\_IsRing(NULL) to be 0, while PostGIS returns NULL.



```
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)
```



ST IsClosed, ST IsSimple, ST StartPoint, ST EndPoint

### 7.4.25 ST\_IsSimple

ST IsSimple —  $\text{IsSimple}(\text{ST})$  TRUE

## Synopsis

boolean **ST\_IsSimple**(geometry geomA);

**Notes**

When the geometry is a **LINESTRING**, **ST\_IsSimple** returns **TRUE** if the geometry is not self-intersecting. For **POLYGON**, **ST\_IsSimple** returns **TRUE** if the geometry is not self-intersecting and is not a **self-intersecting polygon**. **ST\_IsSimple** also checks for **self-intersecting polygons** and **self-intersecting lines**. **ST\_IsSimple** is **OpenGIS compliant** (Ensuring OpenGIS compliance of geometries).



### Note

SQL-MM **ST\_IsSimple**(NULL) returns 0, PostGIS **ST\_IsSimple**(NULL) returns 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.

**Examples**

```
SELECT ST_IsSimple(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))'));
st_issimple
-----
f
(1 row)

SELECT ST_IsSimple(ST_GeomFromText('LINESTRING(1 1,2 2,2 3.5,1 3,1 2,2 1)'));
st_issimple
-----
f
(1 row)
```

**See Also**

**ST\_IsValid**

## 7.4.26 ST\_M

**ST\_M** — Returns the M coordinate of a Point.

## Synopsis

float **ST\_M**(geometry a\_point);

返回

返回 M 坐标。M 坐标 NULL 表示没有 M 坐标。



#### Note

返回 (点) OGC 简单要素实现规范, 提取器 (extractor) 返回的坐标。

- ✓ 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.

返回

```
SELECT ST_M(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_m
-----
      4
(1 row)
```

返回

[ST\\_GeomFromEWKT](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_Z](#)

## 7.4.27 ST\_MemSize

ST\_MemSize — ST\_Geometry 内存大小。

### Synopsis

integer **ST\_MemSize**(geometry geomA);

返回

ST\_Geometry 内存大小。

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` 返回, TOAST 存储的几何体大小。

`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.

☒

```
--Return how much byte space Boston takes up in our Mass data set
SELECT pg_size_pretty(SUM(ST_MemSize(geom))) as totgeomsum,
pg_size_pretty(SUM(CASE WHEN town = 'BOSTON' THEN ST_MemSize(geom) ELSE 0 END)) As bossum,
CAST(SUM(CASE WHEN town = 'BOSTON' THEN ST_MemSize(geom) ELSE 0 END)*1.00 /
      SUM(ST_MemSize(geom))*100 As numeric(10,2)) As perbos
FROM towns;
```

| totgeomsum | bossum | perbos |
|------------|--------|--------|
| -----      | -----  | -----  |
| 1522 kB    | 30 kB  | 1.99   |

```
SELECT ST_MemSize(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'));
---
```

```
73
```

```
--What percentage of our table is taken up by just the geometry
SELECT pg_total_relation_size('public.neighborhoods') As fulltable_size, sum(ST_MemSize(geom)) As geomsize,
sum(ST_MemSize(geom))*1.00/pg_total_relation_size('public.neighborhoods')*100 As pergeom
FROM neighborhoods;
fulltable_size geomsize pergeom
-----
262144 96238 36.71188354492187500000
```

## 7.4.28 ST\_NDims

ST\_NDims — ST\_Geometry ☒☒☒☒☒☒☒☒☒☒.

### Synopsis

integer **ST\_NDims**(geometry g1);

☒☒

☒☒☒☒☒☒☒☒☒☒☒☒☒. PostGIS ☒ 2 - 2 ☒☒ (x,y), 3 - 3 ☒☒ (x,y,z), 3 - ☒☒☒☒☒ 2 ☒☒ (x,y,m), ☒☒☒ 4 - ☒☒☒☒☒ 3 ☒☒☒☒☒ (x,y,z,m) ☒☒☒☒☒☒☒.

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

SQL

```
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
```

SQL

[ST\\_CoordDim](#), [ST\\_Dimension](#), [ST\\_GeomFromEWKT](#)

## 7.4.29 ST\_NPoints

**ST\_NPoints** — Returns the number of points in a geometry.

### Synopsis

integer **ST\_NPoints**(geometry g1);

SQL

Available for: POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION.   
 Available for: 2.0.0 POLYHEDRALSURFACE (polyhedral surface).



### Note

1.3.4 **ST\_NPoints** (curve) returns the number of points in the curve. 1.3.4 **ST\_NPoints** (polyhedral surface) returns the number of vertices.



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



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

SQL

```
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
```

¶¶

ST\_NumPoints

7.4.30 ST\_NRings

ST\_NRings — ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

Synopsis

integer **ST\_NRings**(geometry geomA);

¶¶

¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. NumInteriorRings ¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶.

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.

¶¶

```
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 row) | 1      |  | 0           |

¶¶

ST\_NumInteriorRings

7.4.31 ST\_NumGeometries

ST\_NumGeometries — ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶¶¶.

Synopsis

integer **ST\_NumGeometries**(geometry geom);



Returns the number of elements in a geometry collection (GEOMETRYCOLLECTION or MULTI\*). For non-empty atomic geometries returns 1. For empty geometries returns 0.

Version: 2.0.0, TIN

`__cplusplus`: 2.0.0 if compiled with C++ standard library; otherwise NULL. 2.0.0 if compiled with libstdc++, 1 if compiled with libc++.

- ✔ 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).



```
--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
```



ST GeometryN, ST Multi

### 7.4.32 ST NumInteriorRings

ST NumInteriorRings — □□□□□□□□□□□□□□□□□□□□.

## Synopsis

```
integer ST_NumInteriorRings(geometry a polygon);
```



XXXXXXXXXXXXXXXXXXXX. XXXXXXXXXXXX NULL XXXXXX.

- ✔ This method implements the SQL/MM specification. SQL-MM 3: 8.2.5

☒☒

```
--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;
```

☒☒

[ST\\_NumInteriorRing](#), [ST\\_PointN](#)

### 7.4.33 ST\_NumInteriorRing

**ST\_NumInteriorRing** — 返回多边形内部环的数量。 **ST\_NumInteriorRings** 返回多边形内部环的数量。

#### Synopsis

integer **ST\_NumInteriorRing**(geometry a\_polygon);

☒☒

[ST\\_NumInteriorRings](#), [ST\\_PointN](#)

### 7.4.34 ST\_NumPatches

**ST\_NumPatches** — 返回多边形内部环的数量。 返回多边形内部环的数量。 NULL 返回 NULL。

#### Synopsis

integer **ST\_NumPatches**(geometry g1);

☒☒

返回多边形内部环的数量。 返回多边形内部环的数量。 NULL 返回 NULL。 **ST\_NumGeometries** 返回多边形内部环的数量。 MM 返回多边形内部环的数量。 **ST\_NumGeometries** 返回多边形内部环的数量。

2.0.0 返回多边形内部环的数量。



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.

❏

```
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, 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
```

❏

**ST\_GeomFromEWKT, ST\_NumGeometries**

### 7.4.35 ST\_NumPoints

ST\_NumPoints — ST\_LineString ❏ ST\_CircularString ❏.

#### Synopsis

integer **ST\_NumPoints**(geometry g1);

❏

ST\_LineString ❏ ST\_CircularString ❏. 1.4 ❏. 1.4 ❏, ❏ ST\_NPoints ❏. ❏ ST\_NPoints ❏.

- ✓ 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

❏

```
SELECT ST_NumPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 ←
    29.07)'));
--result
4
```

❏

**ST\_NPoints**

### 7.4.36 ST\_PatchN

ST\_PatchN — ST\_Geometry 国国国国国国国国国国.

#### Synopsis

geometry **ST\_PatchN**(geometry geomA, integer n);

国国

国国 POLYHEDRALSURFACE, POLYHEDRALSURFACEM 国国 1-国 N 国国国 (国) 国国国国国. 国国国国 NULL 国国国国. 国国国国国国国国国 ST\_GeometryN 国国国国国国国国国. ST\_GeometryN 国国国国国国国国国.



#### Note

国国国 1-国国国国.



#### Note

国国国国国国国国国国 ST\_Dump 国国国国国国国国国.

2.0.0 国国国国国国国国国国.



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.

国国

```
--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))
```

国国

**ST\_AsEWKT, ST\_GeomFromEWKT, ST\_Dump, ST\_GeometryN, ST\_NumGeometries**

### 7.4.37 ST\_PointN

ST\_PointN — ST\_LineString 或 ST\_CircularString 的第 N 個點。

#### Synopsis

geometry **ST\_PointN**(geometry a\_linestring, integer n);

返回

如果 N 是負數，則返回第 -N 個點。如果 N 是 0，則返回 NULL。如果 N 是 1，則返回第一個點。如果 N 是 -1，則返回最後一個點。



#### Note

0.8.0 版本實現了 OGC 簡單特徵實現規格 (SQL 1.1)。OGC 簡單特徵實現規格 (SQL 1.1) 實現了 0-索引。0-索引從 0 開始。



#### Note

如果 N 是負數，則返回第 -N 個點。如果 N 是 0，則返回 NULL。如果 N 是 1，則返回第一個點。如果 N 是 -1，則返回最後一個點。

- ✓ 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

PostGIS 2.0.0 版本實現了 OGC 簡單特徵實現規格 (SQL 1.1)。PostGIS 2.0.0 版本實現了 0-索引。0-索引從 0 開始。PostGIS 2.3.0 版本實現了 (-1 索引) 實現。

範例

```
-- 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)
```



## ST\_RemoveRepeatedPoints, ST\_PointN

### 7.4.39 ST\_StartPoint

ST\_StartPoint — Returns the first point of a LineString.

## Synopsis

```
geometry ST_StartPoint(geometry geomA);
```



LINESTRING ☐☐ CIRCULARLINESTRING ☐☐☐☐☐☐☐☐☐ POINT ☐☐☐☐☐☐. ☐☐☐☐☐☐☐  
LINESTRING ☐☐ CIRCULARLINESTRING ☐☐☐☐☐ NULL ☐☐☐☐☐☐.

- ✔ 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.

```

PostGIS 2.0.0
PostGIS 2.0.0
NULL
PostGIS 2.0.0

```



### 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)
```

ST\_LineString 和 ST\_CircularString 返回几何对象。

```
SELECT ST_AsText(ST_StartPoint('CIRCULARSTRING(5 2,-3 1.999999, -2 1, -4 2, 6 3)')::geometry ↵
);
 st_astext
-----
POINT(5 2)
```

和

**ST\_EndPoint, ST\_PointN**

## 7.4.40 ST\_Summary

ST\_Summary — 返回几何对象的摘要字符串。

### Synopsis

```
text ST_Summary(geometry g);
text ST_Summary(geography g);
```

和

返回摘要字符串。  
返回摘要字符串的格式如下：

- M: M 值。
- Z: Z 值。
- B: 边界值。
- G: 几何体 (类型) 值。
- S: 表面值。



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

1.2.2 返回摘要字符串。

版本: 2.0.0 返回摘要字符串。

版本: 2.1.0 和。返回摘要字符串的格式如下 S 值。

版本: 2.2.0 返回 TIN 几何体 (curve) 值。

SQL

```
=# 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;
-----+-----
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)
```

SQL

[PostGIS\\_DropBBox](#), [PostGIS\\_AddBBox](#), [ST\\_Force3DM](#), [ST\\_Force3DZ](#), [ST\\_Force2D](#), [geography](#)  
[ST\\_IsValid](#), [ST\\_IsValid](#), [ST\\_IsValidReason](#), [ST\\_IsValidDetail](#)

### 7.4.41 ST\_X

ST\_X — Returns the X coordinate of a Point.

#### Synopsis

float **ST\_X**(geometry a\_point);

SQL

RETURNS X coordinate. X coordinate NULL if geometry is NULL. If geometry is not a point, an error is raised.



**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.

例

```
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)
```

例

[ST\\_Centroid](#), [ST\\_GeomFromEWKT](#), [ST\\_M](#), [ST\\_XMax](#), [ST\\_XMin](#), [ST\\_Y](#), [ST\\_Z](#)

### 7.4.42 ST\_Y

**ST\_Y** — Returns the Y coordinate of a Point.

#### Synopsis

float **ST\_Y**(geometry a\_point);

例

返回 Y 坐标。Y 坐标为 NULL 时返回 NULL。支持 3D 几何。



#### 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.

例

```
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)
```

☐☐

[ST\\_Centroid](#), [ST\\_GeomFromEWKT](#), [ST\\_M](#), [ST\\_X](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_Z](#)

### 7.4.43 ST\_Z

**ST\_Z** — Returns the Z coordinate of a Point.

#### Synopsis

float **ST\_Z**(geometry a\_point);

☐☐

☐☐☐☐ Z ☐☐☐☐☐☐☐☐. Z ☐☐☐☐☐☐☐☐ NULL ☐☐☐☐☐☐. ☐☐☐☐☐☐☐☐☐☐☐☐.



#### 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.

☐☐

```
SELECT ST_Z(ST_GeomFromEWKT('POINT(1 2 3 4)'));
st_z
-----
3
(1 row)
```

☐☐

[ST\\_GeomFromEWKT](#), [ST\\_M](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 7.4.44 ST\_Zmflag

ST\_Zmflag — ST\_Geometry

#### Synopsis

smallint **ST\_Zmflag**(geometry geomA);

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.

```
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
```

**ST\_CoordDim, ST\_NDims, ST\_Dimension**

### 7.4.45 ST\_HasZ

ST\_HasZ — Checks if a geometry has a Z dimension.

#### Synopsis

boolean **ST\_HasZ**(geometry geom);



Checks if the input geometry has a Z dimension and returns a boolean value. If the geometry has a Z dimension, it returns true; otherwise, it returns false.

Geometry objects with a Z dimension typically represent three-dimensional (3D) geometries, while those without it are two-dimensional (2D) geometries.

This function is useful for determining if a geometry has elevation or height information.

Availability: 3.5.0



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



This function supports M coordinates.



```
SELECT ST_HasZ(ST_GeomFromText('POINT(1 2 3)'));
-- result
true
```

```
SELECT ST_HasZ(ST_GeomFromText('LINESTRING(0 0, 1 1)'));
-- result
false
```



**ST\_Zmflag**

**ST\_HasM**

## 7.4.46 ST\_HasM

**ST\_HasM** — Checks if a geometry has an M (measure) dimension.

### Synopsis

boolean **ST\_HasM**(geometry geom);



Checks if the input geometry has an M (measure) dimension and returns a boolean value. If the geometry has an M dimension, it returns true; otherwise, it returns false.

Geometry objects with an M dimension typically represent measurements or additional data associated with spatial features.

This function is useful for determining if a geometry includes measure information.

Availability: 3.5.0



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



This function supports M coordinates.



❏

```
SELECT ST_HasM(ST_GeomFromText('POINTM(1 2 3)'));
--result
true
```

```
SELECT ST_HasM(ST_GeomFromText('LINESTRING(0 0, 1 1)'));
--result
false
```

❏

**ST\_Zmflag**

**ST\_HasZ**

## 7.5 几何函数 (editor)

### 7.5.1 ST\_AddPoint

ST\_AddPoint — 向 LineString 添加点。

#### Synopsis

geometry **ST\_AddPoint**(geometry linestring, geometry point);  
 geometry **ST\_AddPoint**(geometry linestring, geometry point, integer position = -1);

❏

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.

1.1.0 简体中文。



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

❏

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;
```

---

[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);

---

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](#).

---

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.

---

例

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))
```

例

**`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);

例

“GEOMETRYCOLLECTION(POINT(0 0),LINESTRING(1 1,2 2))” → “MULTILINESTRING((1 1,2 2),POINT(0 0))”.

- 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](#).

2.0.0 中文文档.

中文

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)))
```

中文

[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);

**Geometry**

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.

❌❌

```
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)

```

☒☒

## ST\_LineToCurve

### 7.5.5 ST\_Scroll

ST\_Scroll — Change start point of a closed LineString.

#### Synopsis

geometry **ST\_Scroll**(geometry linestring, geometry point);



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.



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)
```



**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);



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).

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).



例

```
SELECT ST_AsEWKT(ST_FlipCoordinates(GeomFromEWKT('POINT(1 2)')));
      st_asewkt
-----
POINT(2 1)
```

例

**ST\_SwapOrdinates**

### 7.5.7 ST\_Force2D

ST\_Force2D — 将“2 维几何”强制为 2D。

#### Synopsis

geometry **ST\_Force2D**(geometry geomA);

例

“2 维几何”强制为 2D 几何。强制 (OGC 2 维几何) OGC 2 维几何。

2.0.0 强制 (polyhedral surface) 强制。

2.1.0 强制, 2.0.x 强制 ST\_Force\_2D 强制。



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



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

例

```
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))
```

☒☒

**ST\_Force3D**

### 7.5.8 ST\_Force3D

ST\_Force3D — ☒☒☒ XYZ ☒☒☒☒☒☒☒☒. ST\_Force3DZ ☒☒☒☒☒☒.

#### Synopsis

geometry **ST\_Force3D**(geometry geomA, float Zvalue = 0.0);

☒☒

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.

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒ (polyhedral surface) ☒☒☒☒☒☒.

☒☒☒☒: 2.1.0 ☒☒☒☒, ☒ 2.0.x ☒☒☒☒☒☒☒☒☒☒ 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.

☒☒

```
--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))
```

☒☒

**ST\_AsEWKT**, **ST\_Force2D**, **ST\_Force3DM**, **ST\_Force3DZ**

### 7.5.9 ST\_Force3DZ

ST\_Force3DZ — ☒☒☒ XYZ ☒☒☒☒☒☒☒☒.

## Synopsis

geometry **ST\_Force3DZ**(geometry geomA, float Zvalue = 0.0);

**Changes**

Forces the geometries into XYZ mode. If a geometry has no Z component, then a *Zvalue* Z coordinate is tacked on.

**2.0.0** **ST\_Force3DZ** (polyhedral surface) **ST\_Force3DZ**.

**2.1.0** **ST\_Force3DZ**, **2.0.x** **ST\_Force3DZ** **ST\_Force3DZ**.

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.

**Examples**

```
--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))
```

**See Also**

**ST\_AsEWKT**, **ST\_Force2D**, **ST\_Force3DM**, **ST\_Force3D**

## 7.5.10 ST\_Force3DM

**ST\_Force3DM** — **ST\_Force3DM** XYM **ST\_Force3DM**.

## Synopsis

geometry **ST\_Force3DM**(geometry geomA, float Mvalue = 0.0);

☒☒

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

☒☒☒☒: 2.1.0 ☒☒☒☒, ☒ 2.0.x ☒☒☒☒☒☒☒☒☒☒ ST\_Force\_3DM ☒☒☒☒☒.

Changed: 3.1.0. Added support for supplying a non-zero M value.



This method supports Circular Strings and Curves.

☒☒

```
--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))
```

☒☒

**ST\_AsEWKT, ST\_Force2D, ST\_Force3DM, ST\_Force3D, ST\_GeomFromEWKT**

### 7.5.11 ST\_Force4D

ST\_Force4D — ☒☒☒ XYZM ☒☒☒☒☒☒☒☒.

#### Synopsis

geometry **ST\_Force4D**(geometry geomA, float Zvalue = 0.0, float Mvalue = 0.0);

☒☒

Forces the geometries into XYZM mode. *Zvalue* and *Mvalue* is tacked on for missing Z and M dimensions, respectively.

☒☒☒☒: 2.1.0 ☒☒☒☒, ☒ 2.0.x ☒☒☒☒☒☒☒☒☒☒ 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.



```
--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))
```



ST AsEWKT, ST Force2D, ST Force3DM, ST Force3D

### 7.5.12 ST\_ForceCollection

ST ForceCollection — .

## Synopsis

```
geometry ST_ForceCollection(geometry geomA);
```



XXXXXXXXXXXXXXXXXXXX. XXXX WKB XXXXXXXXXXXXXXXXXXXX.

2.0.0 (polyhedral surface).

[illegible]

xxxx: 2.1.0 xxxx, x 2.0.x xxxxxxxxxxxxxxx ST Force Collection xxxxx.

- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.



```
SELECT ST_AsEWKT(ST_ForceCollection('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1))'));

```

---

```

                                st_asewkt
-----
GEOMETRYCOLLECTION(POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1)))

```

```
SELECT ST_AsText(ST_ForceCollection('CIRCULARSTRING(220227 150406,2220227 150407,220227 150406)'));

```

---

```

                                st_astext
-----
GEOMETRYCOLLECTION(CIRCULARSTRING(220227 150406,2220227 150407,220227 150406))
(1 row)

```

```
-- POLYHEDRAL example --
SELECT ST_AsEWKT(ST_ForceCollection('POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))'))))

-- st_asewkt
-----
GEOMETRYCOLLECTION(
  POLYGON((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
  POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
  POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
  POLYGON((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
  POLYGON((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
  POLYGON((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
)
```



ST AsEWKT, ST Force2D, ST Force3DM, ST Force3D, ST GeomFromEWKT

### 7.5.13 ST ForceCurve

ST ForceCurve — □□□□□□□□, □□□□□□□□□□□□□□□□□□□□ (upcast) □□□.

## Synopsis

```
geometry ST_ForceCurve(geometry g);
```



Figure 10: A diagram illustrating the relationship between various mathematical concepts. The diagram shows a hierarchy where "Multisurface" is at the top, branching down to "Compound curve" and "Surface". "Compound curve" further branches to "Curve" and "Line". "Surface" branches to "Plane" and "Space". "Curve" branches to "Arc" and "Spiral". "Line" branches to "Straight line" and "Curved line". "Plane" branches to "Flat plane" and "Curved plane". "Space" branches to "Euclidean space" and "Non-Euclidean space".

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.

- ✓ This function supports 3d and will not drop the z-index.
- ✓ This method supports Circular Strings and Curves.

☒☒

```
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) |                                                                      |

☒☒

**ST\_LineToCurve**

### 7.5.14 ST\_ForcePolygonCCW

ST\_ForcePolygonCCW — Orients all exterior rings counter-clockwise and all interior rings clockwise.

#### Synopsis

geometry **ST\_ForcePolygonCCW** ( geometry geom );

☒☒

Forces (Multi)Polygons to use a counter-clockwise orientation for their exterior ring, and a clockwise orientation for their interior rings. Non-polygonal geometries are returned unchanged.

2.2.0 ☒☒☒☒☒☒☒☒☒☒.

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

☒☒

**ST\_ForcePolygonCW** , **ST\_IsPolygonCCW** , **ST\_IsPolygonCW**

### 7.5.15 ST\_ForcePolygonCW

ST\_ForcePolygonCW — Orients all exterior rings clockwise and all interior rings counter-clockwise.

## Synopsis

geometry **ST\_ForcePolygonCW** ( geometry geom );

ⓘ

Forces (Multi)Polygons to use a clockwise orientation for their exterior ring, and a counter-clockwise orientation for their interior rings. Non-polygonal geometries are returned unchanged.

2.2.0 文档.

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

ⓘ

**ST\_ForcePolygonCCW** , **ST\_IsPolygonCCW** , **ST\_IsPolygonCW**

## 7.5.16 ST\_ForceSFS

ST\_ForceSFS — 文档 SFS 1.1 文档.

## Synopsis

geometry **ST\_ForceSFS**(geometry geomA);  
geometry **ST\_ForceSFS**(geometry geomA, text version);

ⓘ

- ✔ 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.

## 7.5.17 ST\_ForceRHR

ST\_ForceRHR — 文档 (orientation) 文档 (Right-Hand Rule) 文档.

## Synopsis

geometry **ST\_ForceRHR**(geometry g);



☒☒

Forces the orientation of the vertices in a polygon to follow a Right-Hand-Rule, in which the area that is bounded by the polygon is to the right of the boundary. In particular, the exterior ring is orientated in a clockwise direction and the interior rings in a counter-clockwise direction. This function is a synonym for **ST\_ForcePolygonCW**



#### Note

The above definition of the Right-Hand-Rule conflicts with definitions used in other contexts. To avoid confusion, it is recommended to use ST\_ForcePolygonCW.

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒ (polyhedral surface) ☒☒☒☒☒☒.



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



This function supports Polyhedral surfaces.

☒☒

```
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)                                                      |           |

☒☒

**ST\_ForcePolygonCCW** , **ST\_ForcePolygonCW** , **ST\_IsPolygonCCW** , **ST\_IsPolygonCW** , **ST\_BuildArea**, **ST\_Polygonize**, **ST\_Reverse**

## 7.5.18 ST\_LineExtend

**ST\_LineExtend** — Returns a line extended forwards and backwards by specified distances.

### Synopsis

geometry **ST\_LineExtend**(geometry line, float distance\_forward, float distance\_backward=0.0);

☒☒

Returns a line extended forwards and backwards by adding new start (and end) points at the given distance(s). A distance of zero does not add a point. Only non-negative distances are allowed. The direction(s) of the added point(s) is determined by the first (and last) two distinct points of the line. Duplicate points are ignored.

Availability: 3.4.0

**Example: Extends a line 5 units forward and 6 units backward**

```
SELECT ST_AsText(ST_LineExtend('LINESTRING(0 0, 0 10)'::geometry, 5, 6));
-----
LINESTRING(0 -6,0 0,0 10,0 15)
```



## ST\_LineSubstring, ST\_LocateAlong, ST\_Project

### 7.5.19 ST\_LineToCurve

ST\_LineToCurve — Converts a linear geometry to a curved geometry.

## Synopsis

```
geometry ST_LineToCurve(geometry geomANoncircular);
```



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.



```
-- 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;
```

| curvedtext                                                                                                                                                                                                                                                                                                                                                              | non_curvedastext |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| CURVEPOLYGON(CIRCULARSTRING(4 3,3.12132034355964 0.878679656440359,   POLYGON((4 ↵<br>3,3.94235584120969 2.41472903395162,3.77163859753386 1.85194970290473,<br>1 0,-1.12132034355965 5.12132034355963,4 3))   3.49440883690764 ↵<br>1.33328930094119,3.12132034355964 0.878679656440359,<br>  2.66671069905881 ↵<br>0.505591163092366,2.14805029<br>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 2.999999999999999 3,3 3 3) | LINESTRING Z (3 3 3,2.4142135623731 ↵
1.58578643762691 3,1 1 3,                               | -0.414213562373092 1.5857864376269 ↵
3,-1 2.999999999999999 3,                               | -0.414213562373101 4.41421356237309 ↵
3,                                                       | 0.999999999999999 5 ↵
3,2.41421356237309 4.4142135623731 ↵
3,3 3 3)

(1 row)
```

ST\_CurveToLine

7.5.20 ST\_Multi

ST\_Multi —

Synopsis

geometry **ST\_Multi**(geometry geom);

Returns the geometry as a MULTI\* geometry collection. If the geometry is already a collection, it is returned unchanged.

例

```
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)))
```

例

**ST\_AsText**

### 7.5.21 ST\_Normalize

ST\_Normalize — 规范化几何对象。

#### Synopsis

geometry **ST\_Normalize**(geometry geom);

例

规范化几何对象。规范化几何对象，规范化几何对象。  
规范化几何对象，规范化几何对象 (规范化几何对象)。  
2.3.0 规范化几何对象。

例

```
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)
```

例

**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);
```

**Notes**

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.

2.0.0 **Changes**

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)
```

**See also**

[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 );

⌘

**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 efficiently.



### 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.

⌘

```
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     | 010100000005f9a72083cdd5e405f9a72083cdd5e40   | POINT(123.456789123456 123.456789123456) |
| 14     | 010100000005f9a72083cdd5e405f9a72083cdd5e40   | POINT(123.456789123456 123.456789123456) |
| 13     | 010100000005f9a72083cdd5e405f9a72083cdd5e40   | POINT(123.456789123456 123.456789123456) |
| 12     | 010100000005c9a72083cdd5e405c9a72083cdd5e40   | POINT(123.456789123456 123.456789123456) |
| 11     | 01010000000409a72083cdd5e40409a72083cdd5e40   | 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      | 0101000000000000083cdd5e400000000083cdd5e40   | POINT(123.456789016724 123.456789016724) |
| 4      | 0101000000000000003cdd5e400000000003cdd5e40   | POINT(123.456787109375 123.456787109375) |
| 3      | 010100000000000000003cdd5e400000000003cdd5e40 | POINT(123.456787109375 123.456787109375) |
| 2      | 0101000000000000000038dd5e4000000000038dd5e40 | POINT(123.45654296875 123.45654296875)   |
| 1      | 01010000000000000000dd5e40000000000dd5e40     | POINT(123.453125 123.453125)             |
| 0      | 01010000000000000000dc5e40000000000dc5e40     | POINT(123.4375 123.4375)                 |
| -1     | 01010000000000000000c05e400000000000c05e40    | POINT(123 123)                           |
| -2     | 0101000000000000000005e400000000000005e40     | POINT(120 120)                           |
| -3     | 01010000000000000000058400000000000005840     | POINT(96 96)                             |
| -4     | 01010000000000000000058400000000000005840     | POINT(96 96)                             |
| -5     | 01010000000000000000058400000000000005840     | POINT(96 96)                             |
| -6     | 01010000000000000000058400000000000005840     | POINT(96 96)                             |
| -7     | 01010000000000000000058400000000000005840     | POINT(96 96)                             |
| -8     | 01010000000000000000058400000000000005840     | POINT(96 96)                             |
| -9     | 01010000000000000000058400000000000005840     | POINT(96 96)                             |
| -10    | 01010000000000000000058400000000000005840     | POINT(96 96)                             |
| -11    | 01010000000000000000058400000000000005840     | POINT(96 96)                             |
| -12    | 01010000000000000000058400000000000005840     | POINT(96 96)                             |
| -13    | 01010000000000000000058400000000000005840     | POINT(96 96)                             |
| -14    | 01010000000000000000058400000000000005840     | POINT(96 96)                             |
| -15    | 01010000000000000000058400000000000005840     | POINT(96 96)                             |

☒☒

ST\_SnapToGrid

### 7.5.24 ST\_RemovePoint

ST\_RemovePoint — Remove a point from a linestring.

#### Synopsis

geometry **ST\_RemovePoint**(geometry linestring, integer offset);

**Notes**

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

1.1.0 **ST\_RemovePoint** was added.



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

**Warning**

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);
```

**See Also**

[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 = 0.0);

**Notes**

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 a non-zero *tolerance* parameter is provided, vertices within the tolerance distance of one another are considered to be duplicates. The distance is computed in 2D (XY plane).

Enhanced: 3.2.0



## 2.2.0 新增功能.

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

❏

```
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)
```

❏

## ST\_Simplify

### 7.5.26 ST\_RemoveIrrelevantPointsForView

**ST\_RemoveIrrelevantPointsForView** — Removes points that are irrelevant for rendering a specific rectangular view of a geometry.

#### Synopsis

geometry **ST\_RemoveIrrelevantPointsForView**(geometry geom, box2d bounds, boolean cartesian\_hint = false);

❏

Returns a **geometry** without points being irrelevant for rendering the geometry within a given rectangular view.

This function can be used to quickly preprocess geometries that should be rendered only within certain bounds.

Only geometries of type (MULTI)POLYGON and (MULTI)LINESTRING are evaluated. Other geometries keep unchanged.

In contrast to `ST_ClipByBox2D()` this function

- sorts out points without computing new intersection points which avoids rounding errors and usually increases performance,
- returns a geometry with equal or similar point number,
- leads to the same rendering result within the specified view, and
- may introduce self-intersections which would make the resulting geometry invalid (see example below).

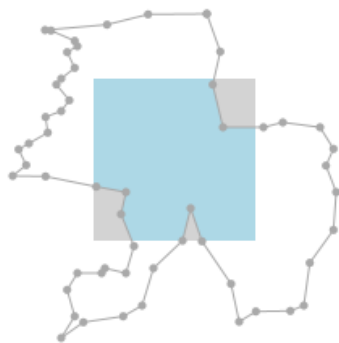
If `cartesian_hint` is set to `true`, the algorithm applies additional optimizations involving cartesian math to further reduce the resulting point number. Please note that using this option might introduce rendering artifacts if the resulting coordinates are projected into another (non-cartesian) coordinate system before rendering.



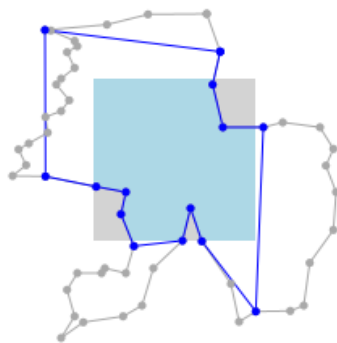
### Warning

For polygons, this function does currently not ensure that the result is valid. This situation can be checked with `ST_IsValid` and repaired with `ST_MakeValid`.

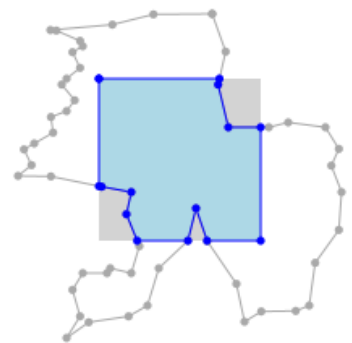
original  
55 points



`ST_RemoveIrrelevantPointsForView(geom, bbox)`  
15 points

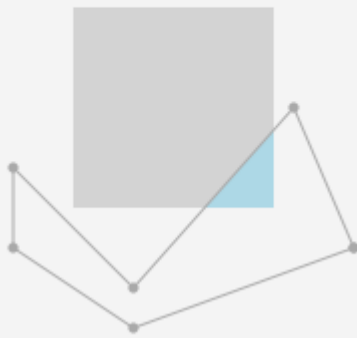


`ST_ClipByBox2D(geom, bbox)`  
15 points

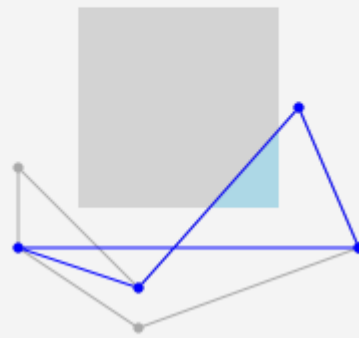


Example: `ST_RemoveIrrelevantPointsForView()` applied to a polygon. Blue points remain, the rendering result (light-blue area) within the grey view box remains as well.

original  
7 points



`ST_RemoveIrrelevantPointsForView(geom, bbox)`  
5 points



Example: Due to the fact that points are just sorted out and no new points are computed, the result of `ST_RemoveIrrelevantPointsForView()` may contain self-intersections.

Availability: 3.5.0

国国

```
SELECT ST_AsText(
    ST_RemoveIrrelevantPointsForView(
        ST_GeomFromText('MULTIPOLYGON(((10 10, 20 10, 30 10, 40 10, 20 20, 10 20, 10 10)),((10 10, 20 10, 20 20, 10 20, 10 10)))'),
        ST_MakeEnvelope(12,12,18,18), true));

st_astext
-----
MULTIPOLYGON(((10 10,40 10,20 20,10 20,10 10)),((10 10,20 10,20 20,10 20,10 10)))
```

```
SELECT ST_AsText(
    ST_RemoveIrrelevantPointsForView(
        ST_GeomFromText('MULTILINESTRING((0 0, 10 0,20 0,30 0), (0 15, 5 15, 10 15, 15 15, 20 15, 25 15, 30 15, 40 15), (13 13,15 15,17 17))'),
        ST_MakeEnvelope(12,12,18,18), true));

st_astext
-----
MULTILINESTRING((10 15,15 15,20 15),(13 13,15 15,17 17))
```

```
SELECT ST_AsText(
    ST_RemoveIrrelevantPointsForView(
        ST_GeomFromText('LINESTRING(0 0, 10 0,20 0,30 0)'),
        ST_MakeEnvelope(12,12,18,18), true));

st_astext
-----
LINESTRING EMPTY
```

```
SELECT ST_AsText(
    ST_RemoveIrrelevantPointsForView(
        ST_GeomFromText('POLYGON((0 30, 15 30, 30 30, 30 0, 0 0, 0 30))'),
        ST_MakeEnvelope(12,12,18,18), true));

st_astext
-----
POLYGON((15 30,30 0,0 0,15 30))
```

```
SELECT ST_AsText(
    ST_RemoveIrrelevantPointsForView(
        ST_GeomFromText('POLYGON((0 30, 15 30, 30 30, 30 0, 0 0, 0 30))'),
        ST_MakeEnvelope(12,12,18,18));

st_astext
-----
POLYGON((0 30,30 30,30 0,0 0,0 30))
```

国国

**ST\_ClipByBox2D, ST\_Intersection**

### 7.5.27 ST\_RemoveSmallParts

ST\_RemoveSmallParts — Removes small parts (polygon rings or linestrings) of a geometry.

#### Synopsis

geometry **ST\_RemoveSmallParts**(geometry geom, double precision minSizeX, double precision minSizeY);

返回

Returns a **geometry** without small parts (exterior or interior polygon rings, or linestrings).

This function can be used as preprocessing step for creating simplified maps, e. g. to remove small islands or holes.

It evaluates only geometries of type (MULTI)POLYGON and (MULTI)LINESTRING. Other geometries remain unchanged.

If *minSizeX* is greater than 0, parts are sorted out if their width is smaller than *minSizeX*.

If *minSizeY* is greater than 0, parts are sorted out if their height is smaller than *minSizeY*.

Both *minSizeX* and *minSizeY* are measured in coordinate system units of the geometry.

For polygon types, evaluation is done separately for each ring which can lead to one of the following results:

- the original geometry,
- a POLYGON with all rings with less vertices,
- a POLYGON with a reduced number of interior rings (having possibly less vertices),
- a POLYGON EMPTY, or
- a MULTIPOLYGON with a reduced number of polygons (having possibly less interior rings or vertices), or
- a MULTIPOLYGON EMPTY.

For linestring types, evaluation is done for each linestring which can lead to one of the following results:

- the original geometry,
  - a LINESTRING with a reduced number of vertices,
  - a LINESTRING EMPTY,
  - a MULTILINESTRING with a reduced number of linestrings (having possibly less vertices), or
  - a MULTILINESTRING EMPTY.
-



*Example: ST\_RemoveSmallParts() applied to a multi-polygon. Blue parts remain.*

Availability: 3.5.0



```
SELECT ST_AsText(
    ST_RemoveSmallParts(
        ST_GeomFromText('MULTIPOLYGON(
            ((60 160, 120 160, 120 220, 60 220, 60 160), (70 170, 70
                210, 110 210, 110 170, 70 170)),
            ((85 75, 155 75, 155 145, 85 145, 85 75)),
            ((50 110, 70 110, 70 130, 50 130, 50 110)))'),
            50, 50));

st_astext
-----
MULTIPOLYGON(((60 160,120 160,120 220,60 220,60 160)),((85 75,155
75,155 145,85 145,85 75)))
```

```
SELECT ST_AsText(
    ST_RemoveSmallParts(
        ST_GeomFromText('LINESTRING(10 10, 20 20)'),
        50, 50));

st_astext
-----
LINESTRING EMPTY
```

### 7.5.28 ST Reverse

ST Reverse — .

## Synopsis

```
geometry ST_Reverse(geometry g1);
```

☒☒

☒☒☒☒☒☒☒☒☒☒, ☒☒☒☒☒☒☒☒☒☒☒☒.

Enhanced: 2.4.0 support for curves was introduced.



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



This function supports Polyhedral surfaces.

☒☒

```
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)
```

## 7.5.29 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);

☒☒

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.

### 1.2.2 `ST_Segmentize`.

Enhanced: 3.0.0 `Segmentize` geometry now produces equal-length subsegments

Enhanced: 2.3.0 `Segmentize` geography now produces equal-length subsegments

`ST_Segmentize`: 2.1.0 `ST_Segmentize` geography now produces equal-length subsegments.

Changed: 2.1.0 As a result of the introduction of geography support, the usage `ST_Segmentize('LINESTRING(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)` )

### `ST_Segmentize`

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:

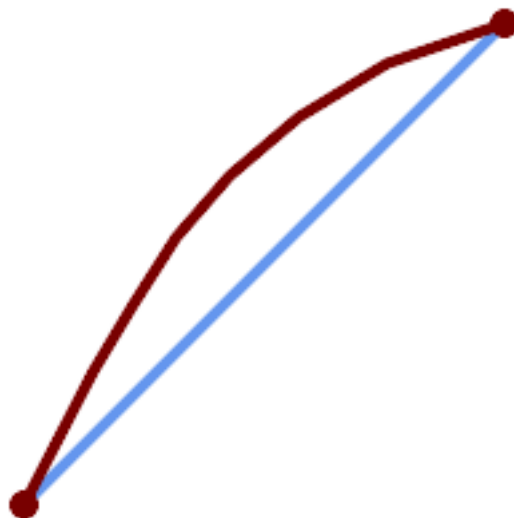
```
SELECT ST_AsText(
  ST_Segmentize(('POLYGON((0 0, 0 8, 30 0, 0 0))'::geometry), 10));
```

```
-----
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*



## ST LineSubstring

### 7.5.30 ST\_SetPoint

ST\_SetPoint — 00000000000000000000000000000000.

## Synopsis

```
geometry ST_SetPoint(geometry linestring, integer zerobasedposition, geometry point);
```



$\{x_n\}_{n=0}^{\infty}$  N- $\{x_n\}_{n=0}^{\infty}$ .  $x_0 = 0$ ,  $x_{n+1} = x_n + \frac{1}{n+1}$ .

1.1.0 ☒☒☒☒☒☒☒☒☒☒☒.

□□□□□□: 2.3.0 □□□□□□□□□□□□□□□□□□□□.



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



```
--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)
```



ST AddPoint, ST NPoints, ST NumPoints, ST PointN, ST RemovePoint



### 7.5.31 ST\_ShiftLongitude

ST\_ShiftLongitude — Shifts the longitude coordinates of a geometry between -180..180 and 0..360.

#### Synopsis

geometry **ST\_ShiftLongitude**(geometry geom);

**Notes**

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

1.3.4 **Warning**: The function ST\_ShiftLongitude is not supported for geometries of type POLYHEDRAL SURFACE. 1.3.4 **Warning**: The function ST\_ShiftLongitude is not supported for geometries of type TIN.



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

**Compatibility**: 2.0.0 **Warning**: (polyhedral surface) **Warning**: TIN **Warning**.

**Compatibility**: 2.2.0 **Warning**, **Warning** "ST\_Shift\_Longitude" **Warning**.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**Notes**

```
--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)
```

☐☐

**ST\_WrapX**

### 7.5.32 ST\_WrapX

ST\_WrapX — X 坐标的环绕。

#### Synopsis

geometry **ST\_WrapX**(geometry geom, float8 wrap, float8 move);

☐☐

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

ST\_WrapX 函数在 GEOS 3.10.0 之前是不可用的。

Availability: 2.3.0 requires GEOS



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

☐☐

```
-- 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);
```

☐☐

**ST\_ShiftLongitude**

### 7.5.33 ST\_SnapToGrid

ST\_SnapToGrid — 將幾何圖形 (geom) 對齊 (snap) 到網格。

#### 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);
```

說明

參數 1, 2, 3: 將幾何圖形 (geom) 對齊 (snap) 到網格。 網格大小由 sizeX, sizeY 指定。 如果 sizeX 或 sizeY 為 NULL，則使用默認值 1.0。 如果 size 為 NULL，則使用默認值 1.0。 如果 pointOrigin 為 NULL，則使用 (0,0)。

參數 4: 1.1.0 版本新增。 將幾何圖形 (geom) 對齊 (snap) 到網格。 網格大小由 sizeX, sizeY, sizeZ, sizeM 指定。 如果 sizeX, sizeY, sizeZ, sizeM 為 0，則使用默認值 1.0。



#### Note

如果 ST\_IsSimple 返回 true，則返回的幾何圖形是簡單的。



#### Note

1.1.0 版本新增。 將幾何圖形 (geom) 對齊 (snap) 到網格。 網格大小由 sizeX, sizeY, sizeZ, sizeM 指定。 1.1.0 版本新增，將幾何圖形 (geom) 對齊 (snap) 到網格。 網格大小由 sizeX, sizeY, sizeZ, sizeM 指定。 如果 sizeX, sizeY, sizeZ, sizeM 為 0，則使用默認值 1.0。

1.0.0RC1 版本新增。

1.1.0 版本新增 Z 和 M 座標。



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

說明

```
--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)

```



ST Snap, ST AsEWKT, ST AsText, ST GeomFromText, ST GeomFromEWKT, ST Simplify

### 7.5.34 ST\_Snap

ST Snap — .

## Synopsis

```
geometry ST_Snap(geometry input, geometry reference, float tolerance);
```



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.

[illegible]

1. 在模型训练过程中，我们采用了多种启发式方法（heuristics）来优化模型的训练效率和性能。这些方法包括：



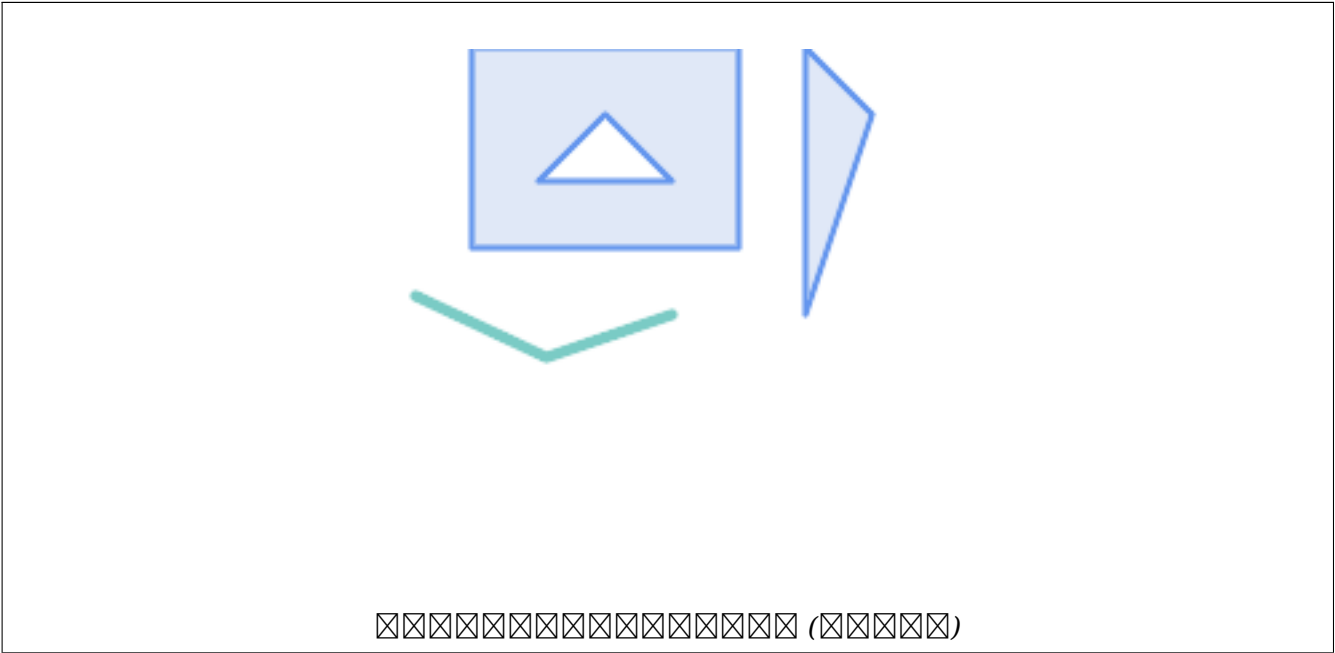
### Note

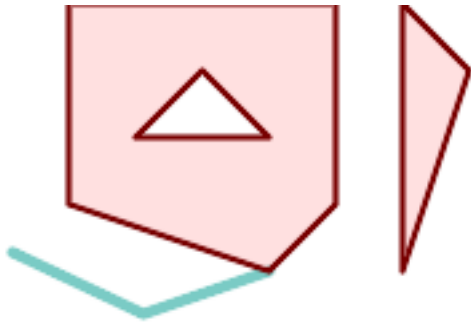
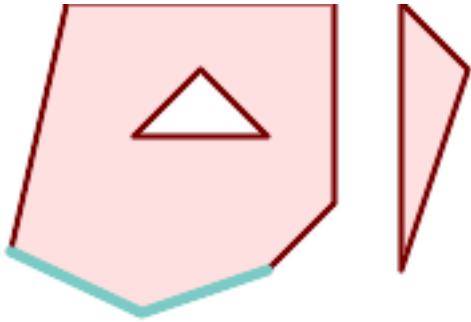
[illegible]

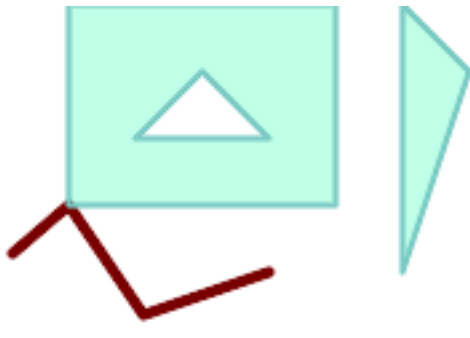
GEOS ☒ ☒ ☒ ☒ ☒

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.





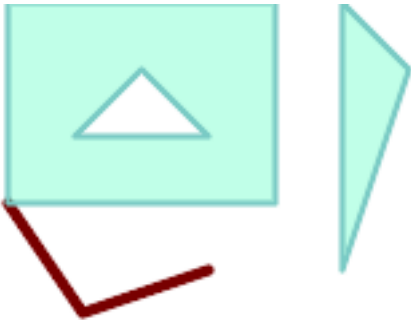
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                 |
| <p>距離 1.01 從多邊形邊界到線段的最短距離。<br/>距離 1.25 從多邊形邊界到線段的最短距離。</p>                                                                                                                                                                                                                                                                                                                                                                                                        | <p>距離 1.25 從多邊形邊界到線段的最短距離。<br/>距離 1.25 從多邊形邊界到線段的最短距離。</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> | <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> |
| <p>polysnapped</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                | <p>polysnapped</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <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>                                                                                                                                                                                                                                                                                                               |



距离 1.01 将红线条移动到与青色多边形边界接触。图中展示了移动前后的两种状态。

```
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)
```



距离 1.25 将红线条移动到与青色多边形边界接触。图中展示了移动前后的两种状态。

```
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)
```

☒

**ST\_SnapToGrid**

**7.5.35 ST\_SwapOrdinates**

ST\_SwapOrdinates — 交换几何体中的纵坐标。

**Synopsis**

geometry **ST\_SwapOrdinates**(geometry geom, cstring ords);

例

ST\_AsText(ST\_SwapOrdinates(ST\_Scale(ST\_SwapOrdinates(g, 'xm'), 2, 1), 'xm'))

ords は ST\_SwapOrdinates の結果を 2 次元に変換する。戻り値は x, y, z, m の順で返す。  
2.2.0 以降は ST\_SwapOrdinates を使用してください。

- ✓ 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).

例

```
-- 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)
```

例

**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);
```



☒☒

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.

GEOS ☒☒☒☒☒

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.

☒☒

```
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
```

☒☒

[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);

## ST\_IsValidDetail

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.

GEOS 2.0.0

2.0.0

## Example

```
--First 3 Rejects from a successful quintuplet experiment
SELECT gid, reason(ST_IsValidDetail(geom)), ST_AsText(location(ST_IsValidDetail(geom))) as
    location
FROM
(SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As geom, gid
FROM (SELECT ST_Buffer(ST_Point(x1*10,y1), z1) As buff, x1*10 + y1*100 + z1*1000 As gid
      FROM generate_series(-4,6) x1
      CROSS JOIN generate_series(2,5) y1
      CROSS JOIN generate_series(1,8) z1
      WHERE x1
> y1*0.5 AND z1 < x1*y1) As e
      INNER JOIN (SELECT ST_Translate(ST_ExteriorRing(ST_Buffer(ST_Point(x1*10,y1), z1)),
        y1*1, z1*2) As line
      FROM generate_series(-3,6) x1
      CROSS JOIN generate_series(2,5) y1
      CROSS JOIN generate_series(1,10) z1
      WHERE x1
> y1*0.75 AND z1 < x1*y1) As f
ON (ST_Area(e.buff)
> 78 AND ST_Contains(e.buff, f.line))
GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(geom) = false
ORDER BY gid
LIMIT 3;
```

| gid  | reason            | location    |
|------|-------------------|-------------|
| 5330 | Self-intersection | POINT(32 5) |
| 5340 | Self-intersection | POINT(42 5) |
| 5350 | Self-intersection | POINT(52 5) |

```
--simple example
SELECT * FROM ST_IsValidDetail('LINESTRING(220227 150406,2220227 150407,222020 150410)');
```

| valid | reason | location |
|-------|--------|----------|
| t     |        |          |

☒☒

**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);

☒☒

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**.

GEOS ☒☒☒☒☒

Availability: 1.4

Availability: 2.0 version taking flags.

☒☒

```
-- invalid bow-tie polygon
SELECT ST_IsValidReason(
  'POLYGON ((100 200, 100 100, 200 200,
    200 100, 100 200))'::geometry) as validity_info;
validity_info
-----
Self-intersection[150 150]
```

```
--First 3 Rejects from a successful quintuplet experiment
SELECT gid, ST_IsValidReason(geom)as validity_info
FROM
(SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As geom, gid
FROM (SELECT ST_Buffer(ST_Point(x1*10,y1), z1) As buff, x1*10 + y1*100 + z1*1000 As gid
      FROM generate_series(-4,6) x1
      CROSS JOIN generate_series(2,5) y1
      CROSS JOIN generate_series(1,8) z1
      WHERE x1
> y1*0.5 AND z1 < x1*y1) As e
      INNER JOIN (SELECT ST_Translate(ST_ExteriorRing(ST_Buffer(ST_Point(x1*10,y1), z1)),
        y1*1, z1*2) As line
      FROM generate_series(-3,6) x1
      CROSS JOIN generate_series(2,5) y1
      CROSS JOIN generate_series(1,10) z1
      WHERE x1
> y1*0.75 AND z1 < x1*y1) As f
ON (ST_Area(e.buff)
> 78 AND ST_Contains(e.buff, f.line))
```

```
GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(geom) = false
ORDER BY gid
LIMIT 3;
```

| gid  | validity_info            |
|------|--------------------------|
| 5330 | Self-intersection [32 5] |
| 5340 | Self-intersection [42 5] |
| 5350 | Self-intersection [52 5] |

```
--simple example
SELECT ST_IsValidReason('LINESTRING(220227 150406,2220227 150407,222020 150410)');
```

| st_isvalidreason |
|------------------|
| Valid Geometry   |

☒☒

**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);
```

☒☒

The function attempts to create a valid representation of a given invalid geometry without losing any of the input vertices. Valid geometries are returned unchanged.

Supported inputs are: POINTS, MULTIPOINTS, LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS and GEOMETRYCOLLECTIONS containing any mix of them.

In case of full or partial dimensional collapses, the output geometry may be a collection of lower-to-equal dimension geometries, or a geometry of lower dimension.

Single polygons may become multi-geometries in case of self-intersections.

The params argument can be used to supply an options string to select the method to use for building valid geometry. The options string is in the format "method=linework|structure keepcollapsed=true|false". If no "params" argument is provided, the "linework" algorithm will be used as the default.

The "method" key has two values.

- "linework" is the original algorithm, and builds valid geometries by first extracting all lines, noding that linework together, then building a value output from the linework.
- "structure" is an algorithm that distinguishes between interior and exterior rings, building new geometry by unioning exterior rings, and then differencing all interior rings.

The "keepcollapsed" key is only valid for the "structure" algorithm, and takes a value of "true" or "false". When set to "false", geometry components that collapse to a lower dimensionality, for example a one-point linestring would be dropped.

GEOS

2.0.0

Enhanced: 2.0.1, speed improvements

Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT.

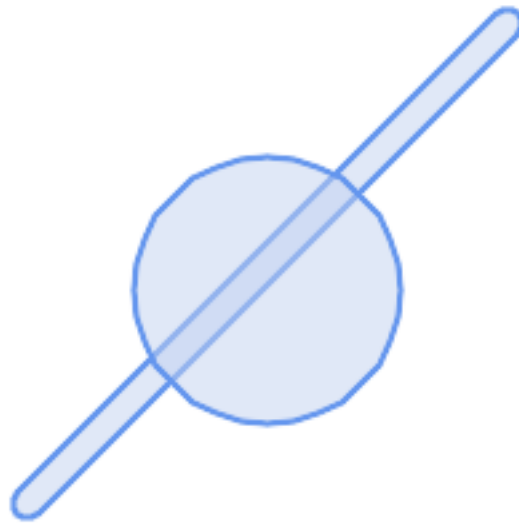
Enhanced: 3.1.0, added removal of Coordinates with NaN values.

Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS >= 3.10.0.

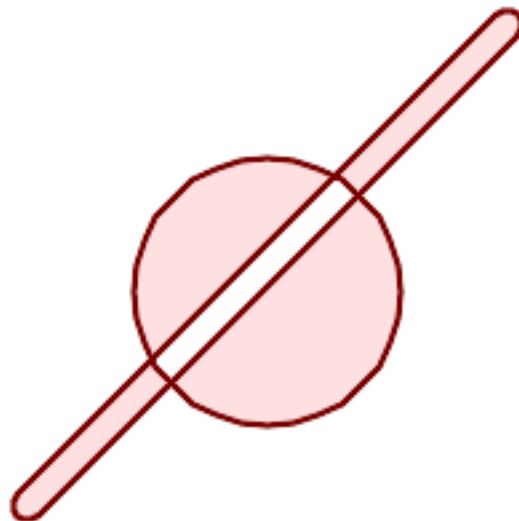


This function supports 3d and will not drop the z-index.

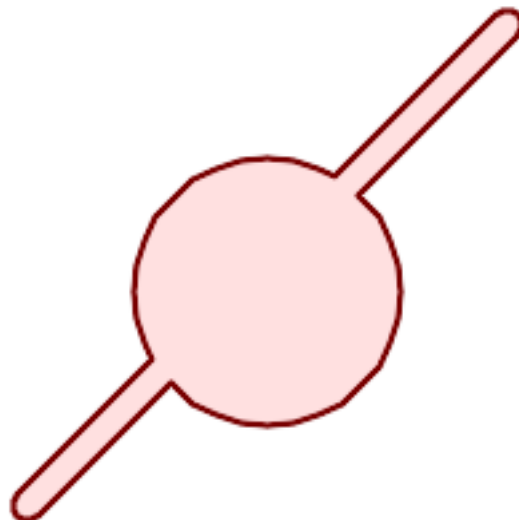
---



*before\_geom: MULTIPOLYGON of 2 overlapping polygons*



*after\_geom: MULTIPOLYGON of 4 non-overlapping polygons*

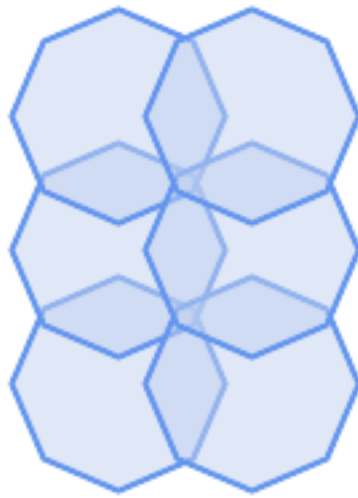


*after\_geom\_structure: MULTIPOLYGON of 1 non-overlapping polygon*

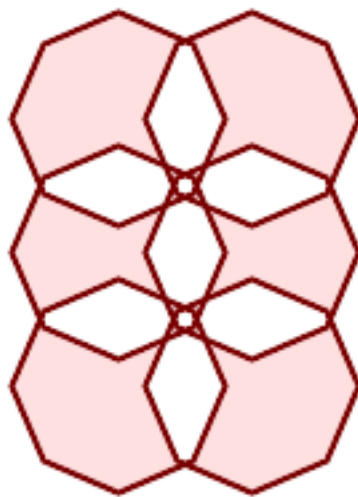
```
SELECT f.geom AS before_geom, ST_MakeValid(f.geom) AS after_geom, ST_MakeValid(f.geom,
  'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,
```

---

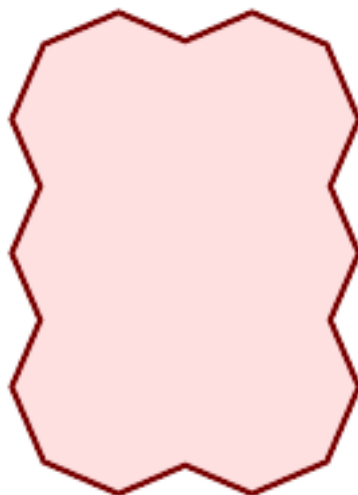
---



*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 ↵
```



---

```

--

```

```

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

```

```

--

```

[ST\\_IsValid](#), [ST\\_Collect](#), [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);

```

--

```

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.

---

☒☒

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;

-----
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;

-----
POINT(143.00000635638918 -36.999986706128176)
(1 row)
```

☒☒

**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);

☒☒

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.

❏❏

-- 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)
```

❏❏

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);

❏❏

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.

❏❏

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
-- result
4326
```

¶¶

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);
```

¶¶

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

1.3.4 [ST\\_MakeEnvelope](#) (curve) [ST\\_MakeEnvelope](#). 1.3.4 [ST\\_MakeEnvelope](#).

¶¶¶: 2.0.0 [ST\\_MakeEnvelope](#) (polyhedral surface) [ST\\_MakeEnvelope](#).

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.

☒

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;
```

☒☒

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);
```

☒☒

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 axis swap steps.
- Concatenated operations of the form: `urn:ogc:def:coordinateOperation,coordinateOperation:EPSG:`

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.

☒☒

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 48.999999999999999)
(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)
```

☒☒

**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);

☒☒

Returns a set of all auth\_srid for the given auth\_name.

Availability: 3.4.0

Proj version 6+

☒☒

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
```

☒☒

**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);

⌘

Returns a metadata record for the requested `auth_srid` for the given `auth_name`. The record will have the `auth_name`, `auth_srid`, `sname`, `srttext`, `proj4text`, and the corners of the area of usage, `point_sw` and `point_ne`.

Availability: 3.4.0

Proj version 6+

⌘

Get the metadata for EPSG:3005.

```
SELECT * FROM postgis_srs('EPSG', '3005');
```

```
auth_name | EPSG
auth_srid | 3005
sname     | NAD83 / BC Albers
srttext   | 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
```

⌘

[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);

⌘

Returns a set of all metadata records in the underlying Proj database. The records will have the `auth_name`, `auth_srid`, `sname`, `srttext`, `proj4text`, and the corners of the area of usage, `point_sw` and `point_ne`.

Availability: 3.4.0

Proj version 6+

❏❏

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      |

❏❏

[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 usage that fully contain the bounds parameter.

**Synopsis**

setof record **postgis\_srs\_search**(geometry bounds, text auth\_name=EPSG);

❏❏

Return a set of metadata records for projected coordinate systems that have areas of usage that fully contain the bounds parameter. Each record will have the `auth_name`, `auth_srid`, `sname`, `srtext`, `proj4text`, and the corners of the area of usage, `point_sw` and `point_ne`.

The search only looks for projected coordinate systems, and is intended for users to explore the possible systems that work for the extent of their data.

Availability: 3.4.0

Proj version 6+

❏❏

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;
```





This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2](#)

GEOS 3.10.0

1.1.0 简体中文.

ST

[ST\\_BuildArea](#), [ST\\_BdMPolyFromText](#)

### 7.8.1.2 ST\_BdMPolyFromText

`ST_BdMPolyFromText` — 将 WKT 多边形文本转换为多边形几何体。

#### Synopsis

geometry **ST\_BdMPolyFromText**(text WKT, integer srid);

ST

将 WKT 多边形文本转换为多边形几何体。



#### Note

WKT 多边形文本必须是一个有效的多边形。如果文本不是一个有效的多边形，`ST_BdPolyFromText` 将返回 NULL，而 `PostGIS` 中的 `ST_BuildArea()` 将返回 NULL。



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2](#)

GEOS 3.10.0

1.1.0 简体中文.

ST

[ST\\_BuildArea](#), [ST\\_BdPolyFromText](#)

### 7.8.1.3 ST\_GeogFromText

`ST_GeogFromText` — WKT (EWKT) 地理多边形文本转换为地理几何体。

#### Synopsis

geography **ST\_GeogFromText**(text EWKT);



WKT `POINT(100 100)` WKT `POINT(100 100)` `SRID 4326`  
`POINT(100 100)`. `ST_GeographyFromText` `POINT(100 100)`. `SRID 4326`.



```

--- 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)'));

```



ST\_AsText, ST\_GeographyFromText

#### 7.8.1.4 ST\_GeographyFromText

[illegible]

## Synopsis

```
geography ST_GeographyFromText(text EWKT);
```



WKT `POINT (100 100)`. `SRID 4326`.



ST GeogFromText, ST AsText

#### 7.8.1.5 ST\_GeomCollFromText

**ST\_GeomCollFromText** — Makes a collection Geometry from collection WKT with the given SRID. If SRID is not given, it defaults to 0.

## Synopsis

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

几何

Makes a collection Geometry from the Well-Known-Text (WKT) representation with the given SRID. If SRID is not given, it defaults to 0.

OGC 3.2.6.2 - 几何 SRID (conformance suite) 几何.

WKT (GEOMETRYCOLLECTION) null 几何.



#### Note

WKT 几何, 几何. 几何 ST\_GeomFromText 几何.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2](#)



This method implements the SQL/MM specification.

几何

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

几何

[ST\\_GeomFromText](#), [ST\\_SRID](#)

### 7.8.1.6 ST\_GeomFromEWKT

ST\_GeomFromEWKT — EWKT(Extended Well-Known Text) 几何 ST\_Geometry 几何.

#### Synopsis

geometry **ST\_GeomFromEWKT**(text EWKT);

几何

OGC EWKT(Extended Well-Known Text) 几何 PostGIS ST\_Geometry 几何.



#### Note

EWKT 几何 OGC 几何, SRID(几何) 几何 PostGIS 几何.

几何: 2.0.0 几何 (polyhedral surface) 几何 TIN 几何.



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).

☒☒

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

☒☒

**ST\_AsEWKT, ST\_GeomFromText**

### 7.8.1.7 ST\_GeomFromMARC21

**ST\_GeomFromMARC21** — Takes MARC21/XML geographic data as input and returns a PostGIS geometry object.

#### Synopsis

geometry **ST\_GeomFromMARC21** ( text marcxml );



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.



Converting MARC21/XML geographic data containing a single POINT encoded as hddd.ddddd

```
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 hddmmss

```

      SELECT
      ST_AsText(
        ST_GeomFromMARC21('
          <record xmlns="http://www.loc.gov/MARC21/slim">
            <leader
>01062cem a2200241 a 4500</leader>
            <controlfield tag="001"
> 84696781 </controlfield>
              <datafield tag="034" ind1="1" ind2=" ">
                <subfield code="a"
>a</subfield>
                <subfield code="b"
>50000</subfield>
                <subfield code="d"
>E0130600</subfield>
                <subfield code="e"
>E0133100</subfield>
                <subfield code="f"
>N0523900</subfield>
                <subfield code="g"
>N0522300</subfield>
              </datafield>
            </record>
          <' ));

      st_astext
-----
POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667  ←
          52.38333333333333,13.1 52.38333333333333,13.1 52.65))
(1 row)

```

Converting MARC21/XML geographic data containing a POLYGON and a POINT:

```

      SELECT
      ST_AsText(
        ST_GeomFromMARC21('
          <record xmlns="http://www.loc.gov/MARC21/slim">
            <datafield tag="034" ind1="1" ind2=" ">
              <subfield code="a"
>a</subfield>
              <subfield code="b"
>50000</subfield>
              <subfield code="d"

```

```

>E0130600</subfield>
      <subfield code="e"
>E0133100</subfield>
      <subfield code="f"
>N0523900</subfield>
      <subfield code="g"
>N0522300</subfield>
      </datafield>
      <datafield tag="034" ind1=" " ind2=" ">
        <subfield code="d"
>W004.500000</subfield>
        <subfield code="e"
>W004.500000</subfield>
        <subfield code="f"
>N054.250000</subfield>
        <subfield code="g"
>N054.250000</subfield>
      </datafield>
    </record>
>'));
```

st\_astext ↩

```

-----
GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.516666666666667  ↩
52.65,13.516666666666667 52.38333333333333,13.1 52.38333333333333,13.1  ↩
52.65)),POINT(-4.5 54.25))
(1 row)
```

☒☒

## ST\_AsMARC21

### 7.8.1.8 ST\_GeometryFromText

ST\_GeometryFromText — WKT(Well-Known Text) ☒☒☒☒☒☒ ST\_Geometry ☒☒☒☒☒☒☒. ☒☒☒☒☒ ST\_GeomFromText ☒☒☒☒☒☒☒☒☒.

#### Synopsis

```

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

☒☒



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\_GeomFromText

#### 7.8.1.9 ST\_GeomFromText

ST\_GeomFromText — WKT `TEXT` `ST_Geometry` `GEOMETRY`.

## Synopsis

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



OGC WKT(Well-Known Text)  PostGIS ST\_Geometry .

**Note**  
ST\_Geometry

ST\_GeomFromText (geometry\_text [SRID]) geometry

- ✔ This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**.  
✔ 3.2.6.2 - SRID (conformance suite)
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.40
- ✔ This method supports Circular Strings and Curves.

**Note**  
While ST Po

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**  
 ☒☒☒☒:

Warning: PostGIS 2.0.0 消息: ST\_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') 消息: PostGIS 2.0.0 消息: SQL/MM 消息: ST\_GeomFromText('GEOMETRYCOLLECTION EMPTY') 消息:



```
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)');

```



ST\_GeomFromEWKT, ST\_GeomFromWKB, ST\_SRID

#### 7.8.1.10 ST\_LineFromText

ST\_LineFromText — `SRID WKT`. SRID, `0`.

## Synopsis

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



Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. If WKT passed in is not a LINESTRING, then null is returned.



### Note

OGC 3.2.6.2 - SRID (conformance suite).

**Note**

ST\_GeomFromText 函数接受 WKT 字符串并返回几何对象。ST\_GeomFromText 函数接受 WKT 字符串并返回几何对象。ST\_GeomFromText 函数接受 WKT 字符串并返回几何对象。



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

SQL

```
SELECT ST_LineFromText('LINESTRING(1 2, 3 4)') AS aline, ST_LineFromText('POINT(1 2)') AS
null_return;
aline | null_return
-----|-----
01020000000200000000000000000000F ... | t
```

SQL

## ST\_GeomFromText

### 7.8.1.11 ST\_MLineFromText

ST\_MLineFromText — WKT 字符串转换为 ST\_MultiLineString 几何对象。

#### Synopsis

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

SQL

Makes a Geometry from Well-Known-Text (WKT) with the given SRID. If SRID is not given, it defaults to 0.

OGC 3.2.6.2 - SRID 值 (conformance suite) 值。

WKT 字符串 null 值。

**Note**

WKT 字符串必须有效，否则将返回空值。ST\_GeomFromText 函数接受 WKT 字符串并返回几何对象。



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

国国

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

国国

**ST\_GeomFromText****7.8.1.12 ST\_MPointFromText**

ST\_MPointFromText — Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

**Synopsis**

geometry **ST\_MPointFromText**(text WKT, integer srid);  
 geometry **ST\_MPointFromText**(text WKT);

国国

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

OGC 国国 3.2.6.2 - 国国国 SRID 国国国国国国 (conformance suite) 国国国国国国国国.

WKT 国国国国国国国国国国 null 国国国国国国.

**Note**

国国 WKT 国国国国国国国国国国国国, 国国国国国国国国国国. 国国国国国国国国国国国国国国国国 ST\_GeomFromText 国国国国国国.



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

国国

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

国国

**ST\_GeomFromText****7.8.1.13 ST\_MPolyFromText**

ST\_MPolyFromText — Makes a MultiPolygon Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

## Synopsis

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



Makes a MultiPolygon from WKT with the given SRID. If SRID is not given, it defaults to 0.

OGC  3.2.6.2 -   SRID  (conformance suite)   .

WKT ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.



### Note

```

WKT Polygon((100 100, 100 200, 200 200, 200 100, 100 100)), SRID=4326;
ST_GeomFromText('Polygon((100 100, 100 200, 200 200, 200 100, 100 100))', 4326);

```

- ✔ 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



```
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);
```



ST\_GeomFromText, ST\_SRID

#### 7.8.1.14 ST PointFromText

ST\_PointFromText — `SRID` `WKT` `SRID` `0`.

## Synopsis

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

⌘

Constructs a PostGIS ST\_Geometry point object from the OGC Well-Known text representation. If SRID is not given, it defaults to unknown (currently 0). If geometry is not a WKT point representation, returns null. If completely invalid WKT, then throws an error.



#### Note

ST\_PointFromText expects 2 values, an OGC SRID. If SRID is not given, it defaults to unknown (currently 0). If geometry is not a WKT point representation, returns null. If completely invalid WKT, then throws an error.



#### Note

WKT is a text representation of a geometry. ST\_GeomFromText expects 2 values, an OGC SRID. If SRID is not given, it defaults to unknown (currently 0). If geometry is not a WKT point representation, returns null. If completely invalid WKT, then throws an error.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). 3.2.6.2 - OGC SRID (conformance suite).



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

⌘

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

⌘

[ST\\_GeomFromText](#), [ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_SRID](#)

### 7.8.1.15 ST\_PolygonFromText

ST\_PolygonFromText — Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

#### Synopsis

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

⌘

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. Returns null if WKT is not a polygon.

OGC 3.2.6.2 - OGC SRID (conformance suite).



**Note**

国国 WKT 国国国国国国国国国国国国国国, 国国国国国国国国国国. 国国国国国国国国国国国国国国国国 ST\_GeomFromText 国国国国国国.



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

国国

```
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
-----
0103000000010000000500000006...
```

```
SELECT ST_PolygonFromText('POINT(1 2)') IS NULL as point_is_notpoly;
point_is_not_poly
-----
t
```

国国

**ST\_GeomFromText**

### 7.8.1.16 ST\_WKTToSQL

ST\_WKTToSQL — WKT(Well-Known Text) 国国国国国国 ST\_Geometry 国国国国国国. 国国国 ST\_GeomFromText 国国国国国国国国.

#### Synopsis

geometry **ST\_WKTToSQL**(text WKT);

国国



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

国国

**ST\_GeomFromText**

## 7.8.2 Well-Known Binary (WKB)

### 7.8.2.1 ST\_GeogFromWKB

ST\_GeogFromWKB — WKB 转 EWKB(转 WKB) 转地理几何。

#### Synopsis

geography **ST\_GeogFromWKB**(bytea wkb);

注意

ST\_GeogFromWKB 转 WKB 转 PostGIS 转 WKB 转地理几何。 转 SQL 转 (Geometry Factory) 转。

SRID 转, 转 4326(WGS84 转) 转。



This method supports Circular Strings and Curves.

注意

```
--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)
```

注意

**ST\_GeogFromText**, **ST\_AsBinary**

### 7.8.2.2 ST\_GeomFromEWKB

ST\_GeomFromEWKB — EWKB(Extended Well-Known Binary) 转 ST\_Geometry 转。

#### Synopsis

geometry **ST\_GeomFromEWKB**(bytea EWKB);

EWKB

OGC EWKB(Extended Well-Known Binary) PostGIS ST\_Geometry EWKB.



#### Note

EWKB OGC EWKB, SRID(SRID) PostGIS EWKB.

2.0.0 (polyhedral surface) 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).

EWKB

NAD83 (SRID 4269) LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932) EWKB



#### Note

: (\) (') standard\_conforming\_strings \ " AsEWKB EWKB.

```
SELECT ST_GeomFromEWKB(E'\001\002\000\000 \255\020\000\000\003\000\000\000\344 ←
J=
\013B\312Q\300n\303(\010\036!E@'\277E' 'K
\312Q\300\366{b\235*!E@\225|\354.P\312Q
\300p\231\323e1!E@');
```



#### Note

In PostgreSQL 9.1+ - standard\_conforming\_strings is set to on by default, where as in past versions it was set to off. You can change defaults as needed for a single query or at the database or server level. Below is how you would do it with standard\_conforming\_strings = on. In this case we escape the ' with standard ansi ', but slashes are not escaped

```
set standard_conforming_strings = on;
SELECT ST_GeomFromEWKB('\001\002\000\000 \255\020\000\000\003\000\000\000\344J=\012\013B
\312Q\300n\303(\010\036!E@'\277E' 'K\012\312Q\300\366{b\235*!E@\225|\354.P\312Q\012\300 ←
p\231\323e1')
```

EWKB

ST\_AsBinary, ST\_AsEWKB, ST\_GeomFromWKB

### 7.8.2.3 ST\_GeomFromWKB

ST\_GeomFromWKB — WKB(Well-Known Binary) SRID

#### Synopsis

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

ST\_GeomFromWKB WKB SRID( ID) SQL (Geometry Factory) ST\_WKBToSQL

SRID, 0(unkown)

✓ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2](#) - SRID (conformance suite)

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

✓ This method supports Circular Strings and Curves.

```
--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)
```

ST\_WKBToSQL, ST\_AsBinary, ST\_GeomFromEWKB

### 7.8.2.4 ST\_LineFromWKB

ST\_LineFromWKB — SRID WKB LINESTRING.

#### Synopsis

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

例

ST\_LineFromWKB WKB SRID( ID) - LINESTRING - . SQL (Geometry Factory) . SRID, 0 . bytea, NULL.



#### Note

OGC 3.2.6.2 - SRID (conformance suite).



#### Note

LINESTRING, ST\_GeomFromWKB . ST\_GeomFromWKB, .

✓ 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

例

```
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
```

例

ST\_GeomFromWKB, ST\_LinestringFromWKB

### 7.8.2.5 ST\_LinestringFromWKB

ST\_LinestringFromWKB — SRID WKB.

## Synopsis

geometry **ST\_LinestringFromWKB**(bytea WKB);  
 geometry **ST\_LinestringFromWKB**(bytea WKB, integer srid);

返回

**ST\_LinestringFromWKB** 将 WKB 转换为 SRID(指定 ID) 的 LINESTRING。 - 返回, LINESTRING 返回。 返回 SQL 函数 (Geometry Factory) 返回。

SRID 指定, 返回 0 返回。 返回 bytea 返回 LINESTRING 返回, NULL 返回。



### Note

OGC 3.2.6.2 - 返回 SRID 返回 (conformance suite) 返回。



### Note

返回 LINESTRING 返回, 返回 **ST\_GeomFromWKB** 返回。 返回 **ST\_GeomFromWKB** 返回, LINESTRING 返回。

✓ 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

返回

```
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
```

返回

**ST\_GeomFromWKB, ST\_LineFromWKB**

## 7.8.2.6 ST\_PointFromWKB

**ST\_PointFromWKB** — 返回 SRID 返回 WKB 返回。

## Synopsis

geometry **ST\_GeomFromWKB**(bytea geom);  
 geometry **ST\_GeomFromWKB**(bytea geom, integer srid);

**ST\_GeomFromWKB**

**ST\_PointFromWKB** converts WKB to SRID (SRID ID) geometry - POINT - geometry. SQL Geometry Factory (Geometry Factory) geometry.

SRID geometry, geometry 0 geometry. bytea geometry, NULL geometry.

- ✓ 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\_GeomFromWKB**

```
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)
```

**ST\_GeomFromWKB**

**ST\_GeomFromWKB, ST\_LineFromWKB**

### 7.8.2.7 ST\_WKBToSQL

**ST\_WKBToSQL** — WKB(Well-Known Binary) geometry ST\_Geometry geometry. SRID geometry ST\_GeomFromWKB geometry.

**Synopsis**

geometry **ST\_WKBToSQL**(bytea WKB);

国国



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

国国

**ST\_GeomFromWKB**

**7.8.3 Other Formats****7.8.3.1 ST\_Box2dFromGeoHash**

ST\_Box2dFromGeoHash — GeoHash 国国国国国国 BOX2D 国国国国国国.

**Synopsis**

box2d **ST\_Box2dFromGeoHash**(text geohash, integer precision=full\_precision\_of\_geohash);

国国

GeoHash 国国国国国国 BOX2D 国国国国国国.

If no precision is specified ST\_Box2dFromGeoHash returns a BOX2D based on full precision of the input GeoHash string.

precision 国国国国国国, ST\_Box2dFromGeoHash 国 GeoHash 国国国国国国国国国国国国国国国国 BOX2D 国国国国国国. 国国国国国国国国 BOX2D 国国国国国国国国国国国国国国国国国国国国国国国国国国国国.

2.1.0 国国国国国国国国国国国国.

国国

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0');
```

```
          st_geomfromgeohash
```

```
-----
BOX(-115.172816 36.114646,-115.172816 36.114646)
```

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 0);
```

```
          st_box2dfromgeohash
```

```
-----
BOX(-180 -90,180 90)
```

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10);
```

```
          st_box2dfromgeohash
```

```
-----
BOX(-115.17282128334 36.1146408319473,-115.172810554504 36.1146461963654)
```





ST GeoHash, ST\_GeomFromGeoHash, ST\_PointFromGeoHash

### 7.8.3.2 ST\_GeomFromGeoHash

ST\_GeomFromGeoHash — GeoHash ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

## Synopsis

```
geometry ST_GeomFromGeoHash(text geohash, integer precision=full_precision_of_geohash);
```



GeoHash 1111111111111111. 11111 GeoHash 1111111111111111.

```
precision 0x00000000, ST_GeomFromGeoHash 0x00 GeoHash 0x0000000000000000  
0x00000000.
```

```
precision 0x000000, ST_GeomFromGeoHash 0 GeoHash 0x00000000000000000000000000000000
0x0000.
```

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.



```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0'));
                                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('9qqj7nmxcggy4d0dbxqz0', 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('9qqj7nmxcggy4d0dbxqz0', 10));
                                st_astext
-----
POLYGON((-115.17282128334 36.1146408319473,-115.17282128334 36.1146461963654,-115.172810554504 36.1146461963654,-115.172810554504 36.1146408319473,-115.17282128334 36.1146408319473))
```



ST GeoHash, ST\_Box2dFromGeoHash, ST\_PointFromGeoHash

### 7.8.3.3 ST\_GeomFromGML

ST\_GeomFromGML — 将 GML 几何对象转换为 PostGIS 几何对象。

#### Synopsis

```
geometry ST_GeomFromGML(text geomgml);
geometry ST_GeomFromGML(text geomgml, integer srid);
```

返回

OGC GML 几何对象转换为 PostGIS ST\_Geometry 几何对象。

ST\_GeomFromGML 将 GML 几何对象 (geometry fragment) 转换为 PostGIS 几何对象。该 GML 几何对象必须是有效的。

支持的 OGC GML 几何对象类型:

- GML 3.2.1 几何对象
- GML 3.1.1 几何对象 SF-2 (GML 3.1.0 到 3.0.0 几何对象)
- GML 2.1.2

OGC GML 规范: <http://www.opengeospatial.org/standards/gml>

1.5 版本使用 LibXML2 1.6 版本。

支持的: 2.0.0 版本 (polyhedral surface) 和 TIN 几何对象。

支持的: 2.0.0 版本 SRID 几何对象。



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 几何对象 (MultiGeometry) 支持 2D 和 3D 几何对象。PostGIS 支持 Z 索引的 ST\_GeomFromGML 支持 2D 几何对象。

GML 几何对象 SRS 指定。PostGIS 支持 SRS 指定, 使用 ST\_GeomFromGML 指定 SRS 指定。GML 几何对象 srsName 指定, 使用指定。

ST\_GeomFromGML 支持 GML 几何对象。支持的 GML 几何对象 XLink 几何对象。



#### Note

ST\_GeomFromGML 支持 SQL/MM 几何对象。

**例: srsName 设置**

```
SELECT ST_GeomFromGML($$
  <gml:LineString xmlns:gml="http://www.opengis.net/gml"
    srsName="EPSG:4269">
    <gml:coordinates>
      -71.16028,42.258729 -71.160837,42.259112 -71.161143,42.25932
    </gml:coordinates>
  </gml:LineString>
$$);
```

**例: XLink 设置**

```
SELECT ST_GeomFromGML($$
  <gml:LineString xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    srsName="urn:ogc:def:crs:EPSG::4269">
    <gml:pointProperty>
      <gml:Point gml:id="p1"
        ><gml:pos
        >42.258729 -71.16028</gml:pos
        ></gml:Point>
      </gml:pointProperty>
      <gml:pos
        >42.259112 -71.160837</gml:pos>
      <gml:pointProperty>
        <gml:Point xlink:type="simple" xlink:href="#p1"/>
      </gml:pointProperty>
    </gml:LineString>
$$);
```

**例: 多边形**

```
SELECT ST_AsEWKT(ST_GeomFromGML('
  <gml:PolyhedralSurface xmlns:gml="http://www.opengis.net/gml">
  <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:polygons>
</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)))

```

文档

Section [2.2.3](#), [ST\\_AsGML](#), [ST\\_GMLToSQL](#)

### 7.8.3.4 ST\_GeomFromGeoJSON

ST\_GeomFromGeoJSON — GeoJSON 文档 PostGIS 文档.

#### Synopsis

```

geometry ST_GeomFromGeoJSON(text geomjson);
geometry ST_GeomFromGeoJSON(json geomjson);
geometry ST_GeomFromGeoJSON(jsonb geomjson);

```

GeoJSON is supported by PostGIS.

ST\_GeomFromGML is a JSON (geometry fragment) parser. It takes JSON as input and returns a geometry.

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.

2.0.0 is supported by JSON-C 0.9.



#### Note

JSON-C is a library for parsing JSON, and is used by PostGIS. JSON-C is a library for parsing JSON, and is used by PostGIS. JSON-C is a library for parsing JSON, and is used by PostGIS. Section 2.2.3



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

```
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[-48.23456,20.12345]}')) ←
      As wkt;
wkt
-----
POINT(-48.23456 20.12345)
```

```
-- a 3D linestring
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"LineString","coordinates":
      ":[1,2,3],[4,5,6],[7,8,9]}')) As wkt;
wkt
-----
LINESTRING(1 2,4 5,7 8)
```

ST\_AsText, ST\_AsGeoJSON, Section 2.2.3

### 7.8.3.5 ST\_GeomFromKML

ST\_GeomFromKML — KML parser PostGIS

#### Synopsis

geometry **ST\_GeomFromKML**(text geomkml);

文档

OGC KML 文档 PostGIS ST\_Geometry 文档。

ST\_GeomFromKML 从 KML 文档 (geometry fragment) 文档。 从 KML 文档 文档。

文档 OGC KML 文档:

- KML 2.2.0 文档

OGC KML 文档: <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 文档 SQL/MM 文档。

文档: **srsName** 文档

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

文档

Section [2.2.3, ST\\_AsKML](#)

### 7.8.3.6 ST\_GeomFromTWKB

ST\_GeomFromTWKB — TWKB("Tiny Well-Known Binary") 文档。

#### Synopsis

geometry **ST\_GeomFromTWKB**(bytea twkb);

文档

ST\_GeomFromTWKB 从 TWKB("Tiny Well-Known Binary") 文档 (WKB) 文档。

❏

```
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)
```

❏

**ST\_AsTWKB**

### 7.8.3.7 ST\_GMLToSQL

ST\_GMLToSQL — GML 文档 ST\_Geometry 文档. 文档 ST\_GeomFromGML 文档.

#### Synopsis

geometry **ST\_GMLToSQL**(text geomgml);  
 geometry **ST\_GMLToSQL**(text geomgml, integer srid);

❏

 This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (文档)

1.5 文档. LibXML2 1.6 文档.

文档: 2.0.0 文档 (polyhedral surface) 文档 TIN 文档.

文档: 2.0.0 文档 SRID 文档.

❏

Section 2.2.3, **ST\_GeomFromGML**, **ST\_AsGML**

### 7.8.3.8 ST\_LineFromEncodedPolyline

ST\_LineFromEncodedPolyline — 文档 (polyline) 文档.

#### Synopsis

geometry **ST\_LineFromEncodedPolyline**(text polyline, integer precision=5);

[illegible]

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.

☒☒: <http://developers.google.com/maps/documentation/utilities/polylinealgorithm>

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.



```
-- 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)
```



## ST AsEncodedPolyline

### 7.8.3.9 ST\_PointFromGeoHash

ST PointFromGeoHash — GeoHash 

## Synopsis

point **ST\_PointFromGeoHash**(text geohash, integer precision=full precision of geohash);



GeoHash 11111111111111111111. 11111111 GeoHash 11111111.

[illegible]

```
precision 0x00000000, ST_PointFromGeoHash 0 GeoHash 0x000000000000000000000000  
0x0000.
```

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.





```

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0'));
          st_astext
-----
POINT(-115.172816 36.114646)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 4));
          st_astext
-----
POINT(-115.13671875 36.123046875)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10));
          st_astext
-----
POINT(-115.172815918922 36.1146435141563)

```

￼

**`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

`void` **`ST_FromFlatGeobufToTable`**(text schemaname, text tablename, bytea FlatGeobuf input data);

￼

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);

￼

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 — Returns WKT(Well-Known Text) representation of the geometry prefixed with the SRID.

#### Synopsis

```
text ST_AsEWKT(geometry g1);
text ST_AsEWKT(geometry g1, integer maxdecimaldigits=15);
text ST_AsEWKT(geography g1);
text ST_AsEWKT(geography g1, integer maxdecimaldigits=15);
```

**Warning**

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.

2.0.0: [ST\\_AsText](#), [ST\\_AsBinary](#), [ST\\_AsEWKB](#) TIN support.



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).

❏❏

```
SELECT ST_AsEWKT('0103000020E61000000100000005000000000000
0000000000000000000000000000000000000000000000000000
F03F000000000000F03F000000000000F03F000000000000F03
F0000000000000000000000000000000000000000000000000000'::geometry);

      st_asewkt
-----
SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
(1 row)

SELECT ST_AsEWKT('010800008003000000000000000060 ↵
E30A4100000000785C0241000000000000F03F0000000018
E20A4100000000485F024100000000000000400000000018
E20A4100000000305C024100000000000000840')

--st_asewkt--
CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)
```

❏❏

[ST\\_AsBinary](#), [ST\\_AsEWKB](#), [ST\\_AsText](#), [ST\\_GeomFromEWKT](#)

### 7.9.1.2 ST\_AsText

[ST\\_AsText](#) — [❏❏/❏❏❏❏](#) WKT(Well-Known Text) [❏❏❏❏](#) SRID [❏❏❏❏❏❏❏❏❏❏❏❏](#).

#### Synopsis

```
text ST_AsText(geometry g1);
text ST_AsText(geometry g1, integer maxdecimaldigits = 15);
text ST_AsText(geography g1);
text ST_AsText(geography g1, integer maxdecimaldigits = 15);
```

❏❏

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.

1.5.0 ☒☒☒☒☒☒☒☒☒☒☒☒.

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.



```
SELECT ST_AsText('010300000001000000050000000000000000  
000000000000000000000000000000000000000000000000  
F03F000000000000F03F000000000000F03F000000000000F03  
F00000000000000000000000000000000000000000000000');
```

```
st_astext
-----
POLYGON((0 0,0 1,1 1,1 0,0 0))
```

Full precision output is the default.

```
SELECT ST_AsText('POINT(111.111111 1.111111)');
      st_astext
-----
POINT(111.111111 1.111111)
```

The *maxdecimaldigits* argument can be used to limit output precision.

```
SELECT ST_AsText('POINT(111.111111 1.111111)'), 2);
      st_astext
-----
POINT(111.11 1.11)
```



ST AsBinary, ST AsEWKB, ST AsEWKT, ST GeomFromText

### 7.9.2 Well-Known Binary (WKB)

#### 7.9.2.1 ST AsBinary

**ST\_AsBinary** — Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.

## Synopsis

```
bytea ST_AsBinary(geometry g1);
bytea ST_AsBinary(geometry g1, text NDR_or_XDR);
bytea ST_AsBinary(geography g1);
bytea ST_AsBinary(geography g1, text NDR_or_XDR);
```



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 'NDR' for little-endian; or 'XDR' for big-endian. Supplying unknown arguments will result in little-endian output.

WKB format is useful to read geometry data from the database and maintaining full numeric precision. This avoids the precision rounding that can happen with text formats such as WKT.

To perform the inverse conversion of WKB to PostGIS geometry use `ST_GeomFromWKB`.



### Note

The OGC/ISO WKB format does not include the SRID. To get the EWKB format which does include the SRID use **ST\_AsEWKB**



### Note

The default behavior in PostgreSQL 9.0 has been changed to output bytea in hex encoding. If your GUI tools require the old behavior, then SET bytea\_output='escape' in your database.

Version: 2.0.0, TIN

☐☐☐☐: 2.0.0 ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐.

☐☐☐☐: 2.0.0 ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐.

1.5.0 ☒☒☒☒☒☒☒☒☒☒☒☒.

[illegible]

✔ 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.



[illegible][illegible]

ST AsBinary, ST\_GeomFromEWKB, ST\_SRID

### 7.9.2.3 ST\_AsHEXEWKB

ST AsHEXEWKB —  $\square\square\square\square\square\square$  (NDR)  $\square\square\square\square\square\square$  (XDR)  $\square\square\square\square\square\square$  HEXEWKB ( $\square\square\square$ )  $\square\square\square\square$   
 $\square\square\square\square\square$ .

## Synopsis

```
text ST_AsHEXEWKB(geometry g1, text NDRorXDR);
text ST_AsHEXEWKB(geometry g1);
```



XXXXXXXXXX (NDR) XXXXXXXXX (XDR) XXXXXXXXX HEXEWKB (XXXX) XXXXXXXXXXXXXXXX. XXXXXXXX  
XXXXXXXXXX NDR XXXXXXXX.



### Note

**Note**

1.2.2 ☐☐☐☐☐☐☐☐☐☐☐.

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.



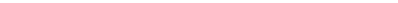
```
SELECT ST_AshExEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
      which gives same answer as

SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326)::text;

st_ashexewkb
-----
0103000020E61000000100000000500
00000000000000000000000000000000
0000000000000000000000000000000F03F
000000000000F03F000000000000F03F000000000000F03
F000000000000000000000000000000000000000000000000
```

### 7.9.3 Other Formats

### 7.9.3.1 ST\_AsEncodedPolyline

ST AsEncodedPolyline — .

## Synopsis

```
text ST_AsEncodedPolyline(geometry geom, integer precision=5);
```



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.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.



```
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`@
```

XXXXXXXXXXXXXXXX (segmentize) XXXXXXXXXXXX, XXXXXXXXXXXX.

```
-- 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;
```

XXXX \$ XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.



```

<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>

```

￼￼

**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);

```

￼￼

Return a FlatGeobuf representation (<http://flatgeobuf.org>) of a set of rows corresponding to a FeatureCollection. NOTE: PostgreSQL bytea cannot exceed 1GB.

row row data with at least a geometry column.

index toggle spatial index creation. Default is false.

geom\_name is the name of the geometry column in the row data. If NULL it will default to the first found geometry column.

Availability: 3.2.0

### 7.9.3.3 ST\_AsGeobuf

ST\_AsGeobuf — Return a Geobuf representation of a set of rows.

#### Synopsis

```

bytea ST_AsGeobuf(anyelement set row);
bytea ST_AsGeobuf(anyelement row, text geom_name);

```



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.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.



```
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
-----
GAAiEAo0CgwIBBoIAAAAAgIAAAE=
```

#### 7.9.3.4 ST\_AsGeoJSON

ST AsGeoJSON — Return a geometry or feature in GeoJSON format.

## Synopsis

```
text ST_AsGeoJSON(record feature, text geom_column="", integer maxdecimaldigits=9, boolean
pretty_bool=false, text id_column="");
text ST_AsGeoJSON(geometry geom, integer maxdecimaldigits=9, integer options=8);
text ST_AsGeoJSON(geography geog, integer maxdecimaldigits=9, integer options=0);
```



Returns a geometry as a GeoJSON "geometry" object, or a row as a GeoJSON "feature" object.

The resulting GeoJSON geometry and feature representations conform with the [GeoJSON specifications RFC 7946](#), except when the parsed geometries are referenced with a CRS other than WGS84 longitude and latitude ([EPSG:4326](#), [urn:ogc:def:crs:OGC::CRS84](#)); the GeoJSON geometry object will then have a short CRS SRID identifier attached by default. 2D and 3D Geometries are both supported. GeoJSON only supports SFS 1.1 geometry types (no curve support for example).

The `geom_column` parameter is used to distinguish between multiple geometry columns. If omitted, the first geometry column in the record will be determined. Conversely, passing the parameter will save column type lookups.

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.



```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [1, 1]
      },
      "id": 1,
      "properties": {
        "name": "one"
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [2, 2]
      },
      "id": 2,
      "properties": {
        "name": "two"
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [3, 3]
      },
      "id": 3,
      "properties": {
        "name": "three"
      }
    }
  ]
}
```

Generate a Feature:

```
SELECT ST_AsGeoJSON(t.*, id_column =
> 'id')
FROM (VALUES (1, 'one', 'POINT(1 1)::geometry)) AS t(id, name, geom);
```

```
st_asgeojson
```

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [1, 1]
  },
  "id": 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],
        [-89.734959999999997, 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]
  ]
}
```

Options argument can be used to add BBOX and CRS in GeoJSON output:

```
SELECT ST_AsGeoJSON(ST_SetSRID('POINT(1 1)::geometry', 4326), 9, 4|1);
```

```
{
  "type": "Point",
  "crs": {
    "type": "name",
    "properties": {
      "name": "urn:ogc:def:crs:EPSG::4326"
    }
  },
  "bbox": [1.000000000, 1.000000000, 1.000000000, 1.000000000],
  "coordinates": [1, 1]
}
```

☒☒

**ST\_GeomFromGeoJSON**, **ST\_ForcePolygonCCW**, **ST\_Transform**

### 7.9.3.5 ST\_AsGML

ST\_AsGML — ☒☒☒ GML 2 ☒☒ GML 3 ☒☒☒☒☒☒☒☒☒☒.

## 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);
```



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  $\times$  2.1.2  $\times \times \times$ , GML 3  $\times$  3.1.1  $\times \times \times \times \times \times \times \times$ .

' 0x' 0x000000 (bitfield) 0x000000. CRS 0x0000 GML 0x00000000, 0x000000/0x0000000000000000.

- 0: GML Short CRS ( $\text{[CRS]}$ : EPSG:4326),  $\text{[SRID]}$
- 1: GML Long CRS ( $\text{[CRS]}$ : urn:ogc:def:crs:EPSG::4326)
- 2: GML 3  $\text{[SRID]}$ ,  $\text{[SRID]}$  srsDimension  $\text{[SRID]}$ .
- 4: GML 3  $\text{[SRID]}$ ,  $\text{[SRID]}$  <Curve>  $\text{[SRID]}$  <LineString>  $\text{[SRID]}$ .
- 16:  $\text{[SRID]}$ / $\text{[SRID]}$  ( $\text{[SRID]}$ : srid=4326)  $\text{[SRID]}$ .  $\text{[SRID]}$ .  $\text{[SRID]}$  (axis order)  $\text{[SRID]}$ , GML 3.1.1  $\text{[SRID]}$ .  $\text{[SRID]}$ ,  $\text{[SRID]}$ .
- 32:  $\text{[SRID]}$  (envelope)  $\text{[SRID]}$ .

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (XXXXXXXX) XXX' XXXXXXXXXXX  
XX' XXXXXXXXXXXX. XXXX NULL XXXXXXXXXX'qml' XXXXXXXX.

1.3.2 ☒☒☒☒☒☒☒☒☒☒☒.

1.5.0 ☒☒☒☒☒☒☒☒☒☒☒☒.

Figure 3: 2.0.0 3D visualization of the GML 3 4' TIN. GML 3 4' TIN 3D visualization. The 3D visualization shows the terrain surface with a 32' grid.

2.0.0 (named arg) 2.0.0

格式: 2.1.0 符合 GML 3 的 ID 格式。



### Note

**Note**

ST AsGML [ ] [ ] [ ] 3 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] TIN [ ] [ ] [ ] [ ] [ ].

- ✔ 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).

## 囧囧: 囧囧 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>
```

## 囧囧: 囧囧 3

```
-- Flip coordinates and output extended EPSG (16 | 1)--
SELECT ST_AsGML(3, ST_GeomFromText('POINT(5.234234233242 6.34534534534)',4326), 5, 17);
  st_asgml
-----
  <gml:Point srsName="urn:ogc:def:crs:EPSG::4326"
><gml:pos
>6.34535 5.23423</gml:pos
></gml:Point>
```

```
-- Output the envelope (32) --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 32);
  st_asgml
-----
  <gml:Envelope srsName="EPSG:4326">
  <gml:lowerCorner
>1 2</gml:lowerCorner>
  <gml:upperCorner
>10 20</gml:upperCorner>
  </gml:Envelope>
```

```
-- Output the envelope (32) , reverse (lat lon instead of lon lat) (16), long srs (1)= 32 | ←
16 | 1 = 49 --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 49);
  st_asgml
-----
  <gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
  <gml:lowerCorner
>2 1</gml:lowerCorner>
  <gml:upperCorner
```

```
>20 10</gml:upperCorner>
</gml:Envelope>
```

```
-- Polyhedral Example --
SELECT ST_AsGML(3, ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0) ←
),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )''));
  st_asgml
-----
<gml:PolyhedralSurface>
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 1 0 0 1 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
```

```
</gml:polygonPatches>
</gml:PolyhedralSurface>
```

ST\_GeomFromGML

7.9.3.6 ST\_AsKML

ST\_AsKML — GML 2 GML 3

Synopsis

```
text ST_AsKML(geometry geom, integer maxdecimaldigits=15, text nprefix=NULL);
text ST_AsKML(geography geog, integer maxdecimaldigits=15, text nprefix=NULL);
```

KML(Keyhole Markup Language) . . . (15), 2



**Warning**  
Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use **ST\_ReducePrecision** with a suitable gridsize first.



**Note**  
PostGIS Proj . Proj **PostGIS\_Full\_Version**.



**Note**  
1.2.2 . 1.3.2 .



**Note**  
: 2.0.0 . .



**Note**  
Changed: 3.0.0 - Removed the "versioned" variant signature



**Note**

ST\_AskML 返回 SRID 4326 的 KML 字符串。



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

例

```
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>

```

例

**ST\_AsSVG, ST\_AsGML**

### 7.9.3.7 ST\_AsLatLonText

ST\_AsLatLonText — 返回指定几何体的经纬度文本。

#### Synopsis

text **ST\_AsLatLonText**(geometry pt, text format=“”);

例

ST\_AsLatLonText(geom, 'WKT').

**Note**

ST\_AsLatLonText 返回的文本格式为“X(经度) Y(纬度) 格式” 格式。经度 (经度 -180 到 180 度, 纬度 -90 到 90 度) 格式。

返回的字符串格式为 "D", "M", "S", (方向, cardinal direction) "C" 格式。D, M, S 格式为 "SSS.SSSS" ("1.0023" 格式)。

M, S, C 格式。"C" 格式为 "SSS.SSSS" 格式。"S" 格式为 "SSS.SSSS" 格式。"M" 格式为 "SSS.SSSS" 格式。

返回的字符串格式为 (方向 0 度) 格式。

2.0 格式。

SQL

SQL

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)'));
      st_aslatlon
-----
2°19'29.928"S 3°14'3.243"W
```

(方向) 格式。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D°M'S.SSS"C'));
      st_aslatlon
-----
2°19'29.928"S 3°14'3.243"W
```

D, M, S, C 格式。格式。

```
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
```

格式。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D°M'S.SSS"));
      st_aslatlon
-----
-2°19'29.928" -3°14'3.243"
```

格式。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
      st_aslatlon
-----
2.3250 degrees S 3.2342 degrees W
```

格式。

```
SELECT (ST_AsLatLonText('POINT (-302.2342342 -792.32498)'));
      st_aslatlon
-----
72°19'29.928"S 57°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' );

☒☒

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): hddmmss
- decimal degrees with cardinal direction: hddd.ddddd
- decimal degrees without cardinal direction: ddd.ddddd
- decimal minutes with cardinal direction: hddmm.mmmm
- decimal minutes without cardinal direction: dddmm.mmmm
- decimal seconds with cardinal direction: hddmmss.sss

The decimal sign may be also a comma, e.g. hddmm,mmm.

The precision of decimal formats can be limited by the number of characters after the decimal sign, e.g. hddmm.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.

☒☒

Converting a POINT to MARC21/XML formatted as hddmmss (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, 'hddmmm.
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>

```

☒☒

**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);

☒☒

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\_TileEnvelope**.

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.

2.2.0



#### 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.

```
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. This assumes the data.geom column has srid of 4326.

```
SELECT ST_AsMVTGeom(
  ST_Transform( geom, 3857 ),
  ST_TileEnvelope(12, 513, 412), extent =
> 4096, buffer =
> 64) AS geom
FROM data
WHERE geom && ST_Transform(ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096)),4326)
```

[ST\\_AsMVT](#), [ST\\_TileEnvelope](#), [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);
```

⊠

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.

2.2.0 `ST_AsMVTGeom`.

⊠

```
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;
```

文档

[ST\\_AsMVTGeom](#), [ST\\_TileEnvelope](#)

### 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);

文档

SVG (Scalar Vector Graphics) 文档。 (relative move) 文档 (absolute move) 文档 0 文档。 (15) 文档。 'rel' 0 文档 cx/cy 文档, 'rel' 1 文档 x/y 文档。 (",") 文档, (";") 文档。

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

文档: 2.0.0 文档 (default arg) 文档 (named arg) 文档。



#### Note

1.2.2 文档 1.4.0 文档 <http://www.w3.org/TR/SVG/paths.html#PathDataBNF> 文档 L 文档。



This method supports Circular Strings and Curves.

文档

```
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 TWKB (Tiny Well-Known Binary) representation.

#### 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);

**Description**

ST\_AsTWKB returns TWKB (Tiny Well-Known Binary) representation. TWKB is a compact binary format for storing and transmitting geometry data. It is a subset of the Well-Known Binary (WKB) format.

ST\_AsTWKB supports all the geometry types supported by PostGIS. It also supports the optional parameters prec, prec\_z, prec\_m, with\_sizes, and with\_boxes. The prec parameter specifies the precision of the geometry. The prec\_z parameter specifies the precision of the Z coordinate. The prec\_m parameter specifies the precision of the M coordinate. The with\_sizes parameter specifies whether to include the size of the geometry in the output. The with\_boxes parameter specifies whether to include the bounding box of the geometry in the output.

ST\_AsTWKB returns a bytea value. The bytea value is a binary representation of the geometry. It can be stored in a database or transmitted over a network. The TWKB format is a compact binary format for storing and transmitting geometry data. It is a subset of the Well-Known Binary (WKB) format.

ST\_AsTWKB supports all the geometry types supported by PostGIS. It also supports the optional parameters prec, prec\_z, prec\_m, with\_sizes, and with\_boxes. The prec parameter specifies the precision of the geometry. The prec\_z parameter specifies the precision of the Z coordinate. The prec\_m parameter specifies the precision of the M coordinate. The with\_sizes parameter specifies whether to include the size of the geometry in the output. The with\_boxes parameter specifies whether to include the bounding box of the geometry in the output.



#### Note

<https://github.com/TWKB/Specification> for more information. <https://github.com/TWKB/twkb.js> for the JavaScript implementation.

Enhanced: 2.4.0 memory and speed improvements.

2.2.0 Added support for geometry collections.



```
SELECT ST_AstWKB('LINESTRING(1 1,5 5)::geometry');
          st_astwkb
```

\x02000202020808

TWKB 格式，`array_agg()` 函数  
 TWKB 格式。

```
SELECT ST_AsTWKB(array_agg(geom), array_agg(gid)) FROM mytable;
```

\x040402020400000202



ST\_GeomFromTWKB, ST\_AsBinary, ST\_AsEWKB, ST\_AsEWKT, ST\_GeomFromText

### 7.9.3.13 ST AsX3D

ST AsX3D — XXX X3D XML XXXXXXX: ISO-IEC-19776-1.2-X3DEncodings-XML XXXXXXX.

## Synopsis

```
text ST_AsX3D(geometry g1, integer maxdecimaldigits=15, integer options=0);
```



<http://www.web3d.org/standards/number/19776-1>
X3D XML
   
 maxdecimaldigits(15)

### Note

[illegible]

' 00' 00000000000000. PostGIS 2.2 0000000, 000 X3D 000000000000000  
0, 0000 x/y 00000000000000000000. 00000 ST\_AsX3D 000000000 (long,lat  
or X,Y) 000000, X3D 00000000/00, y/x 00000000000.

- 0: `X/Y(XXX/XX = X,Y XXX), XXX, X (非XXX (XXX))`.
- 1: `X Y XXXX. XXX (GeoCoordinate) XXXXXXXX, XXXXX "latitude_first"(XXX) XXXXXXXXXXXXXXXX`.
- 2: `XXXXXXXXXXXXX. XXX WGS84 XXX (SRID 4326) XXXX, XXXXXXXX XXXXXXXX. XXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXXX x3D XXX XXXXXXXX. XXXXXXXX GeoCoordinate geoSystem="GD" "WE" "longitude_first" XXXX. X3D XXX GeoCoordinate geoSystem="GD" "WE" "latitude_first" XXXXXXX, (2 + 1) = 3 XXXX`.

| PostGIS 类型                         | 2D X3D 类型                    | 3D X3D 类型                |
|------------------------------------|------------------------------|--------------------------|
| LINESTRING                         | LINESTRING. PolyLine2D       | LineSet                  |
| MULTILINESTRING                    | MULTILINESTRING. PolyLine2D  | IndexedLineSet           |
| MULTIPOINT                         | PointSet                     | PointSet                 |
| POINT                              | POINT. POINT                 | POINT. POINT             |
| (MULTI) POLYGON, POLYHEDRALSURFACE | (MULTI) POLYGON X3D (markup) | IndexedFaceSet (faceset) |
| TIN                                | TriangleSet2D                | IndexedTriangleSet       |



### Note

2. . .   
.

Lots of advancements happening in 3D space particularly with **X3D Integration with HTML5**

FreeWRL is a free, open source, cross platform X3D viewer. It, along with, Free Wrl, is available at <http://freewrl.sourceforge.net/>. FreeWRL\_Launcher is a Java application that runs FreeWRL.

Also check out [PostGIS minimalist X3D viewer](#) that utilizes this function and [x3dDom html/js open source toolkit](#).

2.0.0 0000 ISO-IEC-19776-1.2-X3DEncodings-XML 0000000000.

[illegible]

- ✔ 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).

00: 0000000 X3D 00000000. 00000 FreeWrl 000 X3D 0000000000000000  
 00000000.

```
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>
' ||
ST_AsX3D( ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ) ) ||
'</Shape>
</Transform>
' )
```

```

</Scene>
</X3D>
>' As x3ddoc;

x3ddoc
-----
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d ←
-3.0.dtd">
<X3D>
  <Scene>
    <Transform>
      <Shape>
        <Appearance>
          <Material emissiveColor='0 0 1' />
        </Appearance>
        <IndexedFaceSet coordIndex='0 1 2 3 -1 4 5 6 7 -1 8 9 10 11 -1 12 13 14 15 -1 16 17 ←
18 19 -1 20 21 22 23'>
          <Coordinate point='0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 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*

实验: 实验实验实验 6 实验 3 实验实验实验实验

```

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>

```

### 圖 7.9.3.13: 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>

```

### 圖 7.9.3.14: 多線串 (MULTILINESTRING)

```

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 — 將 GeoHash 轉換為 3D 格式。

## Synopsis

text **ST\_GeoHash**(geometry geom, integer maxchars=full\_precision\_of\_point);

**Notes**

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.

1.4.0 **Changes**



### Note

ST\_GeoHash requires input geometry to be in geographic (lon/lat) coordinates.



This method supports Circular Strings and Curves.

**Examples**

```
SELECT ST_GeoHash( ST_Point(-126,48) );

 st_geohash
-----
c0w3hf1s70w3hf1s70w3

SELECT ST_GeoHash( ST_Point(-126,48), 5);

 st_geohash
-----
c0w3h

-- This line contains the point, so the GeoHash is a prefix of the point code
SELECT ST_GeoHash('LINESTRING(-126 48, -126.1 48.1)::geometry);

 st_geohash
-----
c0w3
```

**See Also**

[ST\\_GeomFromGeoHash](#), [ST\\_PointFromGeoHash](#), [ST\\_Box2dFromGeoHash](#)

## 7.10 (operator)

### 7.10.1 Bounding Box Operators

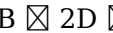
#### 7.10.1.1 &&

**&&** — A  2D  B  2D  TRUE .

#### Synopsis

boolean **&&**( geometry A , geometry B );  
boolean **&&**( geography A , geography B );

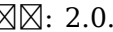


**&&**  A  2D  B  2D  TRUE .



#### Note

 (operand)       .

: 2.0.0  (polyhedral surface) .

1.5.0 .



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



```
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)



**ST\_Intersects**, **ST\_Extent**, **|&>**, **&>**, **&<|**, **&<**, **~**, **@**

#### 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 );

⌘⌘

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.

⌘⌘

```
SELECT ST_Point(1,1) && ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS overlaps;

overlaps
-----
t
(1 row)
```

⌘⌘

**&&(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 );



☒☒

The `&&` operator returns TRUE if the 2D bounding box A intersects the cached 2D bounding box of geometry B, using float precision. This means that if A is a (double precision) `box2d`, it will be internally converted to a float precision 2D bounding box (`BOX2DF`)



#### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

☒☒

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_Point(1,1) AS overlaps;
```

```
overlaps
-----
t
(1 row)
```

☒☒

`&&(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 );

☒☒

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.

SQL

```
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)
```

SQL

**&&(geometry,box2df), &&(box2df,geometry), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @ (geometry,box2df), @ (box2df,geometry), @ (box2df,box2df)**

### 7.10.1.5 &&&

**&&&** — A  $n$  dimensional geometry B  $n$  dimensional geometry TRUE or FALSE.

#### Synopsis

boolean **&&&**( geometry A , geometry B );

SQL

**&&&**  $n$  dimensional geometry A  $n$  dimensional geometry B  $n$  dimensional geometry TRUE or FALSE.



#### Note

**&&&** (operand)  $n$  dimensional geometry.

2.0.0  $n$  dimensional geometry.

- ✓ 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.

**3D Overlaps**

```
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           |

**3DM Overlaps**

```
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           |

**3D Overlaps**

**&&&**

**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 );

**3D Overlaps**

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.

SQL

```
SELECT ST_MakePoint(1,1,1) &&& ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) AS overlaps;
```

```
overlaps
-----
t
(1 row)
```

SQL

```
&&&(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 );

SQL

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 INdices (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.

SQL

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_MakePoint(1,1,1) AS overlaps;
overlaps
-----
t
(1 row)
```

SQL

**`&&&(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 );

SQL

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 INdices (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.

SQL

```
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)
```

SQL

**`&&&(geometry,gidx), &&&(gidx,geometry)`**

**7.10.1.9 &<**

**`&<`** — A `boolean` B `boolean` TRUE `boolean`.

**Synopsis**

boolean **`&<`**( geometry A , geometry B );

SQL

**`&<`** `boolean` A `boolean` B `boolean`, `boolean` B `boolean` TRUE `boolean`.



**Note**

`boolean` (operand) `boolean`.

SQL

```
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)
```

&&, |&>, &>, &<|

7.10.1.10 &<|

&<| — A  $\overline{A \cap B}$   $\overline{A \cap B}$  TRUE  $\overline{A \cap B}$ .

Synopsis

boolean **&<|**( geometry A , geometry B );

**&<|**  $\overline{A \cap B}$   $\overline{A \cap B}$   $\overline{A \cap B}$ ,  $\overline{A \cap B}$  B  $\overline{A \cap B}$   $\overline{A \cap B}$   $\overline{A \cap B}$ , TRUE  $\overline{A \cap B}$ .

- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.



Note

$\overline{A \cap B}$  (operand)  $\overline{A \cap B}$   $\overline{A \cap B}$   $\overline{A \cap B}$   $\overline{A \cap B}$ .

```
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)

&&, |&>, &>, &<

7.10.1.11 &>

&> — A  $\overline{A \cap B}$   $\overline{A \cap B}$   $\overline{A \cap B}$  TRUE  $\overline{A \cap B}$ .

## Synopsis

boolean **&>**( geometry A , geometry B );

描述

**&>** 当 A 与 B 不相交, 返回 B 的几何体是否完全在 A 的几何体内部, TRUE 表示是。



### Note

几何体 (operand) 必须是有效的几何体。

描述

```
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)

描述

**&&**, **|&>**, **&<**, **&<**

## 7.10.1.12 <<

**<<** — A 与 B 不相交 TRUE 表示是。

## Synopsis

boolean **<<**( geometry A , geometry B );

描述

**<<** 当 A 与 B 不相交, 返回 TRUE 表示是。



### Note

几何体 (operand) 必须是有效的几何体。





```

1 | 3 | f
1 | 4 | f
(3 rows)

```

☒☒

<<, >>, |>>

#### 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 );
```

☒☒

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.

☒☒

```

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

```

ST Equals, ST OrderingEquals,  $\sim =$ 

### 7.10.1.15 >>

>> - A ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ B ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ TRUE ☐ ☐ ☐ ☐ ☐.

## Synopsis

```
boolean >>( geometry A , geometry B );
```



>> A B TRUE.



### Note

(operand)

例

```
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)

例

<<, |>>, <<|

### 7.10.1.16 @

@ — B 是否包含 A 的几何体。TRUE 表示包含。

#### Synopsis

boolean @( geometry A , geometry B );

例

@ 是否包含 B 的几何体 A 的几何体。TRUE 表示包含。



#### Note

几何体 (operand) 必须是有效的几何体。

例

```
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)
```

~, &&

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 );

The @ operator returns TRUE if the A geometry’s 2D bounding box is contained the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

✔ This method supports Circular Strings and Curves.

✔ This function supports Polyhedral surfaces.

```
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)
```

&&(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 );
```

⊠

The @ operator returns TRUE if the 2D bounding box A is contained into the B geometry's 2D bounding box, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



#### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

⊠

```
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)
```

⊠

&&(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 );
```



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.



```
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)
```



$\&(\text{geometry}, \text{box2df}), \&\&(\text{box2df}, \text{geometry}), \&\&(\text{box2df}, \text{box2df}), \sim(\text{geometry}, \text{box2df}), \sim(\text{box2df}, \text{geometry}),$   
 $\sim(\text{box2df}, \text{box2df}), @(\text{geometry}, \text{box2df}), @(\text{box2df}, \text{geometry})$

### 7.10.1.20 |&>

|&> - A ☐ ☐ ☐ ☐ ☐ ☐ B ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ TRUE ☐ ☐ ☐ ☐ ☐.

## Synopsis

```
boolean |&>( geometry A , geometry B );
```



```
&> A <- B, TRUE
```



### Note

(operand)

```
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)

&&, &>, &<|, &<

7.10.1.21 |>>

|>> — A  B  TRUE .

Synopsis

boolean |>>( geometry A , geometry B );

The |>> operator returns TRUE if the bounding box of geometry A is strictly above the bounding box of geometry B.



Note

(operand) .

```
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)          |   |       |

<<, >>, <<|

7.10.1.22 ~

~ — A B TRUE .

Synopsis

boolean ~( geometry A , geometry B );

~ A B TRUE .



**Note**  
(operand) .

```
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)
```

@, &&

### 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 );

☒☒

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.

☒☒

```
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)
```

☒☒

&&(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 );



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.



```
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)
```



`&&(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 );
```



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 INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

SQL

```
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)
```

SQL

`&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), @&&(geometry,box2df), @&&(box2df,geometry), @&&(box2df,box2df)`

### 7.10.1.26 ~=

`~=` — A `geometry` B `geometry` TRUE

### Synopsis

boolean `~=`( geometry A , geometry B );

SQL

`~=` `geometry`/ `geometry` A `geometry`/ `geometry` B `geometry` TRUE



#### Note

`geometry` (operand) `geometry` TRUE

1.5.0 `geometry`.



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`.

例

```
select 'LINESTRING(0 0, 1 1)::geometry ~= 'LINESTRING(0 1, 1 0)::geometry as equality;
equality      |
-----+-----
t            |
```

例

`ST_Equals`, `ST_OrderingEquals`, `=`

## 7.10.2 运算符 (operator)

### 7.10.2.1 <->

<-> — A 与 B 的 2 个最近邻居。

#### Synopsis

```
double precision <->( geometry A , geometry B );
double precision <->( geography A , geography B );
```

例

<-> 返回 2 个最近邻居。 "ORDER BY" 子句 (index-assisted) 最近 (nearest neighbor) 邻居。 PostgreSQL 9.5 及以上版本支持 (distance sphere) 距离球 KNN 邻居。



#### Note

操作数 (operand) 必须是 2 个 GiST 索引。 ORDER BY 子句 (index-assisted) 最近 (nearest neighbor) 邻居。



#### Note

例如, 查询 a.geom 的 SRID=3005;POINT(1011102 450541)::geometry 的最近邻居, (子句/CTE(common table expression) 子句) 最近邻居。

OpenGeo workshop: Nearest-Neighbour Searching 研讨会。

2.2.0 版本 -- PostgreSQL 9.5 及以上版本支持 KNN("K nearest neighbor") 最近邻居。 KNN 最近邻居。 PostgreSQL 9.4 及以上版本支持 (distance sphere) 距离球 KNN 最近邻居。

2.2.0 版本 -- PostgreSQL 9.5 及以上版本支持 (Hybrid syntax) 混合语法。 PostgreSQL 2.2 版本, PostgreSQL 9.5 及以上版本支持 (Hybrid syntax) 混合语法。

2.0.0 版本 -- KNN 最近邻居。 PostgreSQL 9.1 及以上版本支持 (distance sphere) 距离球 KNN 最近邻居。

例

```
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)

例 KNN 例:

```
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)

例”EXPLAIN ANALYZE” 例。

PostgreSQL 9.5 例, 例。  
例 KNN 例 CTE(common table expression) 例, 例。  
例:

```
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)

```

繁體中文

**ST\_DWithin, ST\_Distance, <#>**

### 7.10.2.2 |=|

**|=|** — A 與 B 的最接近點 (closest point of approach) 的軌跡 (trajectory)。

#### Synopsis

double precision **|=|**( geometry A , geometry B );

繁體中文

**|=|** 函數 ( **ST\_IsValidTrajectory** 函數 ) 返回 3 個值。第一個值 **ST\_DistanceCPA** 函數，第二個值 **ST\_DistanceCPA** 函數 (PostgreSQL 9.5.0 版本) N 個最近鄰居 (nearest neighbor) 的軌跡。



#### Note

第一個值 (operand) 是 **ST\_IsValidTrajectory** 函數。N 個 GiST 索引。第二個值 **ORDER BY** 函數。第三個值 **ORDER BY** 函數。



#### Note

第二個值 **ST\_DistanceCPA** 函數，第三個值 **a.geom** 函數 **'SRID=3005;LINESTRINGM(0 0 0,0 0 1)::geometry** 函數，(函數/CTE(common table expression) 函數) 函數。

2.2.0 版本。PostgreSQL 9.5 版本 (index-supported) 函數。

繁體中文

```

-- 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)

```



ST DistanceCPA, ST ClosestPointOfApproach, ST IsValidTrajectory

### 7.10.2.3 <#>

$$\langle \# \rangle = A \otimes B \otimes \otimes \otimes \otimes \otimes \otimes \otimes 2 \otimes \otimes \otimes \otimes \otimes \otimes \otimes \otimes \otimes.$$

## Synopsis

```
double precision <#>( geometry A , geometry B );
```



<#> floating point . (PostgreSQL 9.1 )  
 ) . .



### Note

ORDER BY `expression` `ASC` | `DESC` [`NULLS` `FIRST` | `LAST`].



Note!

### Note

```
geom_order_by(ggplot2::ggplot(mtcars) + geom_text(aes(x = gear, y = mpg, label = gear)), ggplot2::geom_text <#> ggplot2::ORDER BY (ST_GeomFromText('POINT(1 2)') <#> geom))
```

2.0.0. PostgreSQL 9.1 KNN.



```
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 |



ST DWithin, ST Distance, &lt;-&gt;

#### 7.10.2.4 <<->>

**<->** — Returns the n-D distance between the A and B geometries or bounding boxes

## Synopsis

```
double precision <<->>( geometry A , geometry B );
```



<<-> □□□□□□□□□□□□□□ N □ (□□□) □□□□□□□. □□□ □□□□□□□  
□□□□□□□□.

### Note

[illegible]

### Note

1. `SELECT a.geom FROM a.geom WHERE SRID=3005;POINT(1011102 450541)::geometry`  
 2. `(SELECT/CTE(common table expression) FROM) FROM`  
 3.

2.2.0 2017年12月15日. PostgreSQL 9.1 2012年12月15日 KNN 2017年12月15日.

☒☒

<->

## 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 );

☒☒

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.



#### Note

Because of floating robustness failures, geometries don't always intersect as you'd expect them to after geometric processing. For example the closest point on a linestring to a geometry may not lie on the linestring. For these kind of issues where a distance of a centimeter you want to just consider as intersecting, use [ST\\_3DDWithin](#).

Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs.

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

SQL

```
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
```

SQL

**ST\_3DDWithin, 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);

SQL

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:  $ST\_Contains(A, B) \Leftrightarrow (A \sqsupset B = B) \wedge (Int(A) \sqcap Int(B) \neq \emptyset)$

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) = true and **ST\_Contains**(B, A) = true, then the two geometries must be topologically equal (**ST\_Equals**(A, B) = true).

**ST\_Contains** is the converse of **ST\_Within**. So, **ST\_Contains**(A, B) = **ST\_Within**(B, A).



#### Note

Because the interiors must have a common point, a subtlety of the definition is that polygons and lines do *not* contain lines and points lying fully in their boundary. For further details see **Subtleties of OGC Covers, Contains, Within**. The **ST\_Covers** predicate provides a more inclusive relationship.

**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Contains`.

GEOS

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.

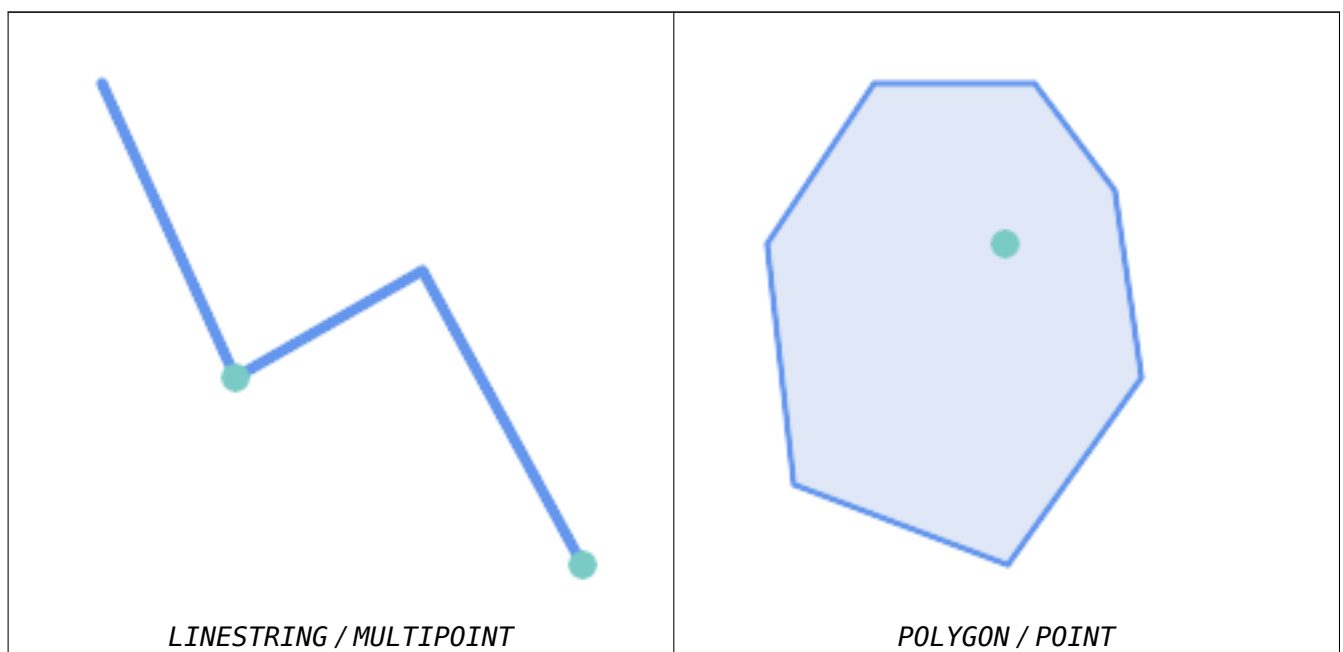


This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3](#) - same as `within(geometry B, geometry A)`



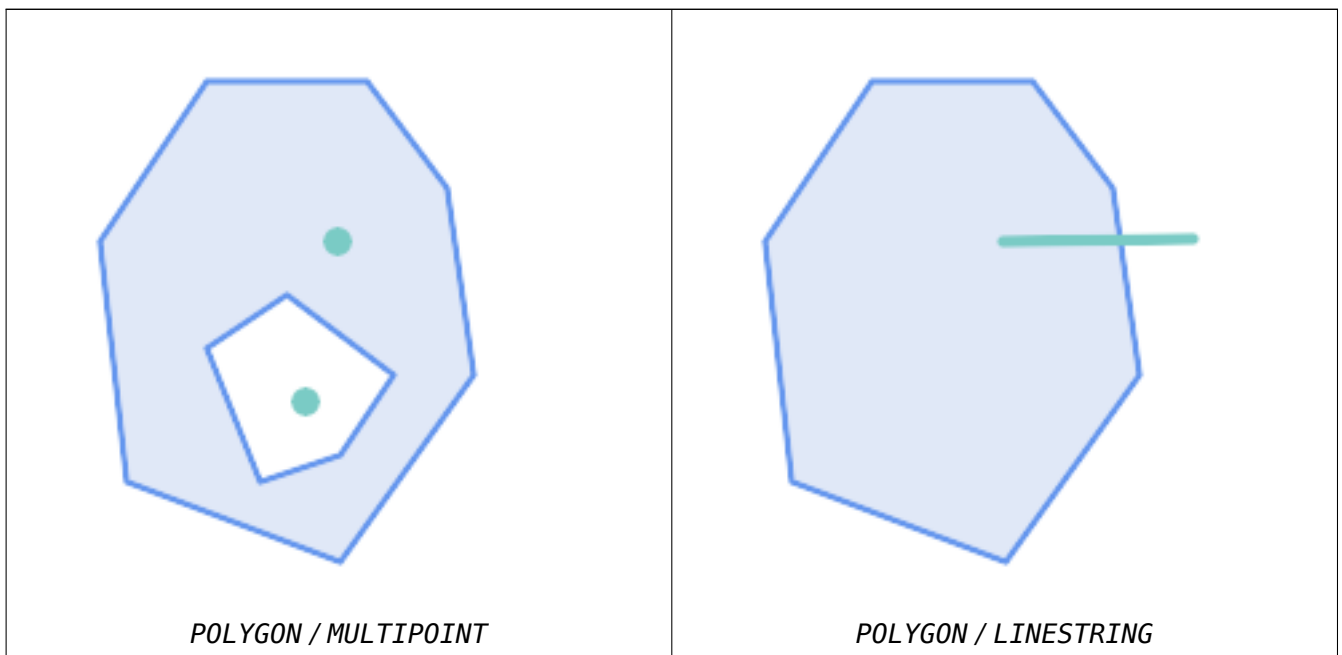
This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

`ST_Contains` returns TRUE in the following situations:

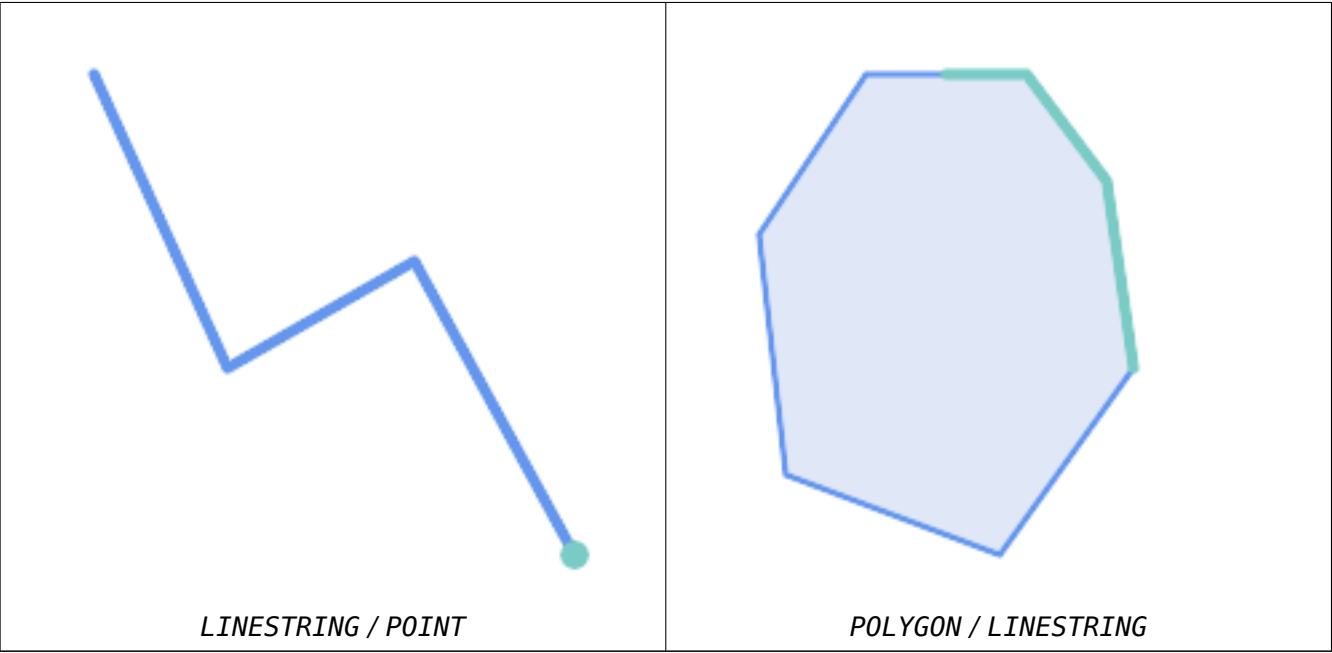




`ST_Contains` returns `FALSE` in the following situations:



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
```

文档

[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);

文档

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:  $ST\_ContainsProperly(A, B) \Leftrightarrow Int(A) \supset B = B$

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.

GEOS 文档

1.4.0 文档.



#### Important

Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`



#### Important

Do not use this function with invalid geometries. You will get unexpected results.



```
--a circle within a circle
SELECT ST_ContainsProperly(smallc, bigc) As smallcontainspropbig,
ST_ContainsProperly(bigc,smallc) As bigcontainspropsmall,
ST_ContainsProperly(bigc, ST_Union(smallc, bigc)) as bigcontainspropunion,
ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
ST_ContainsProperly(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallcontainspropbig | bigcontainspropsmall | bigcontainspropunion | bigisunion |
bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----+-----+-----
f                    | t                    | f                    | t          |
                    | 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
```



`ST_CoveredBy` is the converse of `ST_Covers`. So, `ST_CoveredBy(A,B) = ST_Covers(B,A)`.

Generally this function should be used instead of `ST_Within`, since it has a simpler definition which does not have the quirk that "boundaries are not within their geometry".



**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_CoveredBy`.



**Important**

Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`



**Important**

Do not use this function with invalid geometries. You will get unexpected results.

GEOS

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.

```
--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)
```

`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);
```

国国

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:  $ST\_Covers(A, B) \Leftrightarrow A \sqcup B = B$

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.

GEOS 国国国国国

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.

1.5.0 国国国国国国国国国国国国.

1.2.2 国国国国国国国国国国国国.

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Not an OGC standard, but Oracle has it too.

国国

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
```

**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);

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



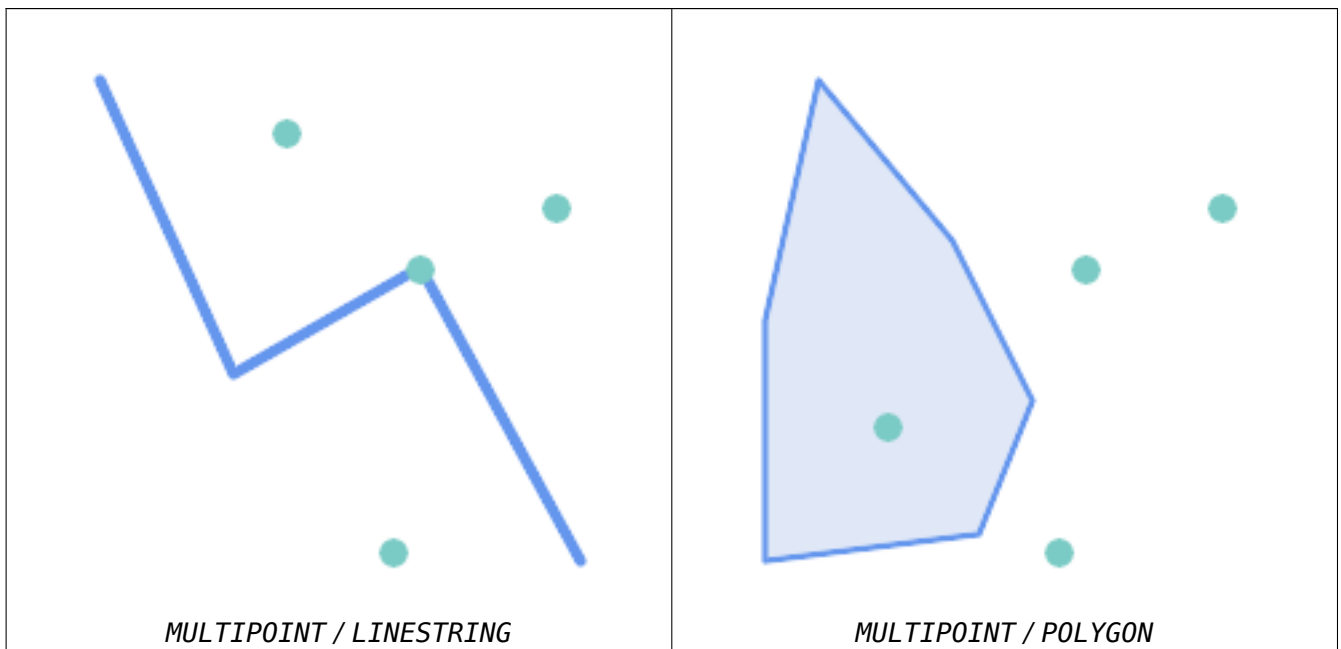
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.13.3](#)

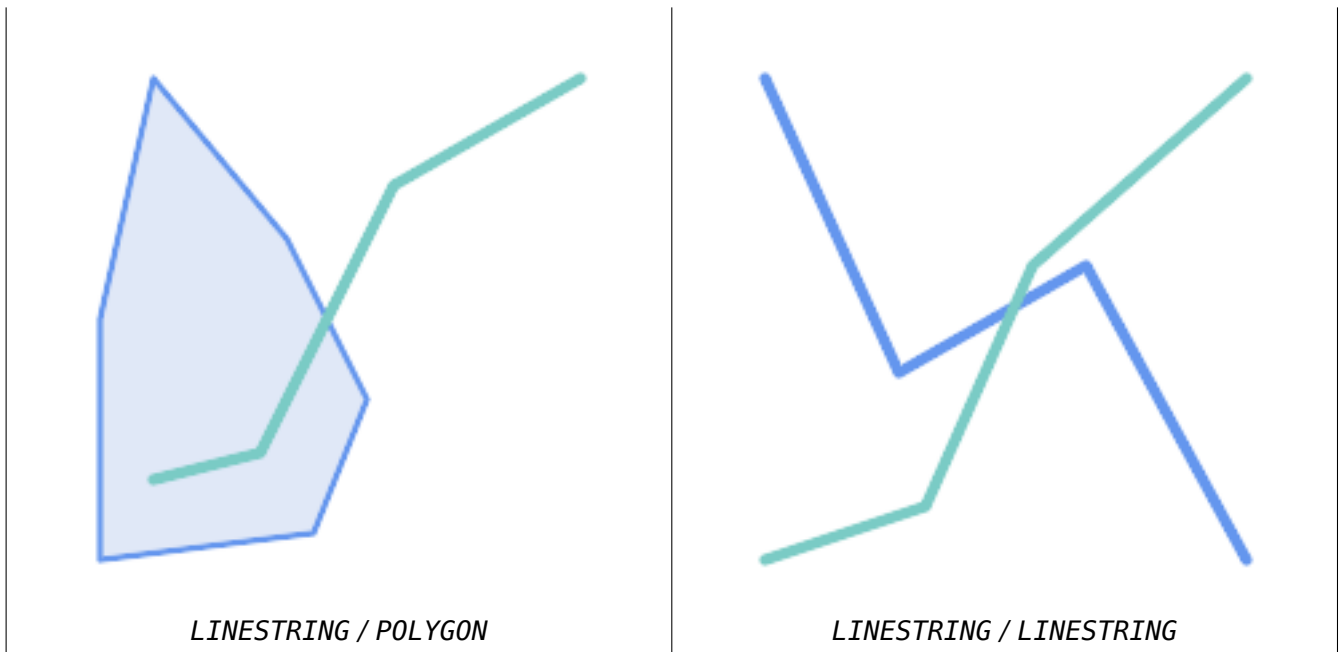


This method implements the SQL/MM specification. SQL-MM 3: 5.1.29



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);
```

☒☒

**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 );

☒☒

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

GEOS ☒☒☒☒☒



### 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

☒☒

```
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)
```

☒☒

**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);

**⚠**

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

**⚠**

```
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)
```

**⚠**

**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 );
```

**⚠**

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).



SQL

```
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)
```

SQL

```
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
```

SQL

&&, [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);

SQL

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.

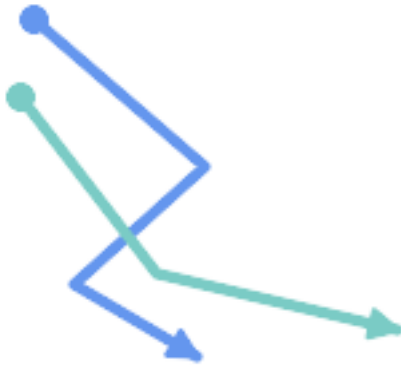
The crossing number has the following meaning:

- 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

**Example:** LINE CROSS LEFT and LINE CROSS RIGHT

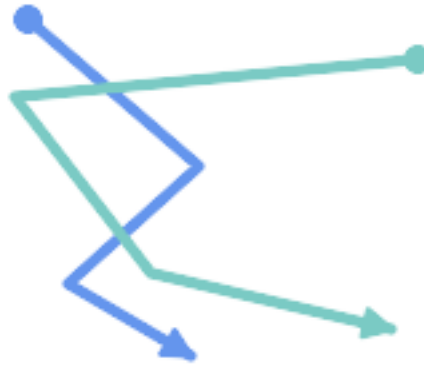


Blue: Line A; Green: Line B

```
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

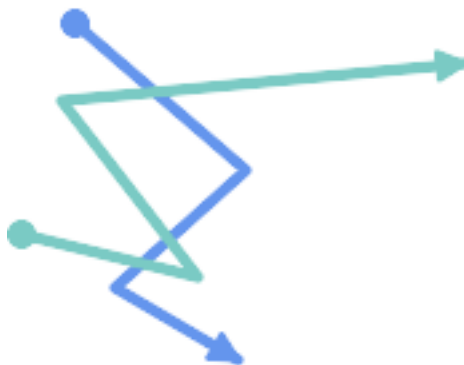


Blue: Line A; Green: Line B

```
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;
```

☒☒

## 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);

☒☒

**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/arcsgde/9.1/sql\\_api/sqlapi3.htm#ST\\_OrderingEquals](http://edndoc.esri.com/arcsgde/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals)



This method implements the SQL/MM specification. SQL-MM 3: 5.1.43

☒☒

```
SELECT ST_OrderingEquals(
  'LINESTRING(0 0, 10 10)',
  'LINESTRING(0 0, 5 5, 10 10)');
```

```
st_orderingequals
-----
f
```

```
SELECT ST_OrderingEquals(
```

```
'LINESTRING(0 0, 10 10)',
'LINESTRING(0 0, 10 10)');

st_orderingequals
-----
t

SELECT ST_OrderingEquals(
  'POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))',
  'POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))');

st_orderingequals
-----
f
```

☒☒

&&, [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);

☒☒

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:  $ST\_Overlaps(A, B) \Leftrightarrow (dim(A) = dim(B) = dim(Int(A) \cap Int(B))) \wedge (A \not\subset B \wedge B \not\subset A)$



#### 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`.

GEOS ☒☒☒☒☒



#### Important

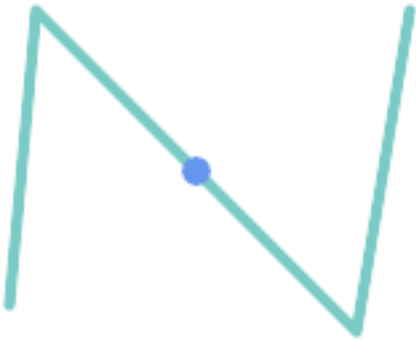
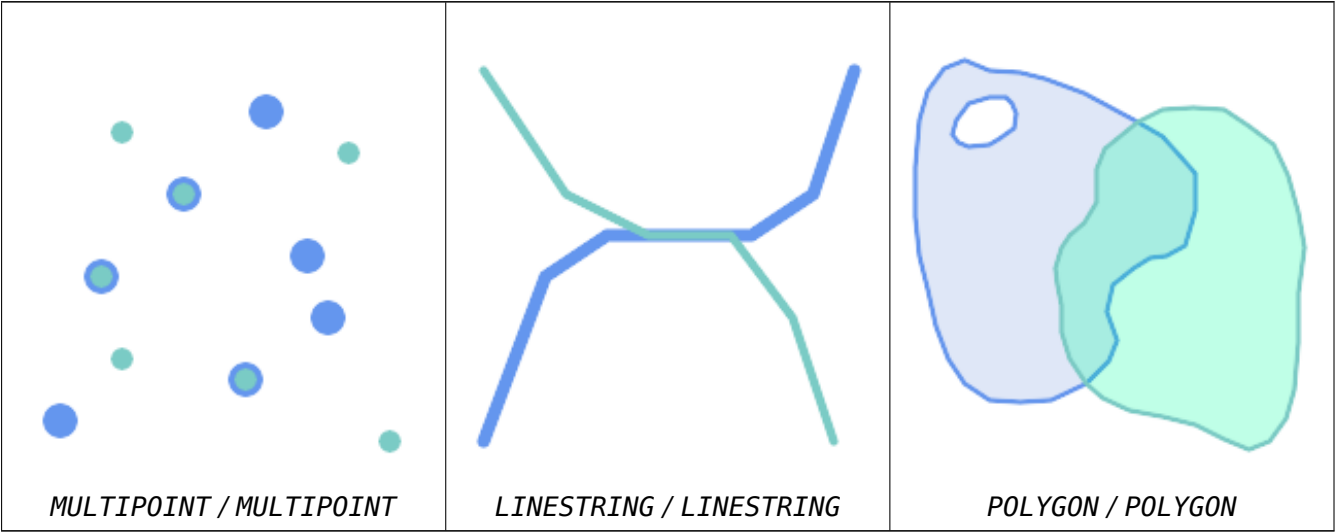
Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

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

🔍🔍

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       |

[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);
```

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

GEOS

Enhanced: 2.0.0 - added support for specifying boundary node rule.



### Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

Using the boolean-valued function to test spatial relationships.

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer( 'POINT(1 2)', 2), '0FFFFF212');
st_relate
-----
t

SELECT ST_Relate(POINT(1 2)', ST_Buffer( 'POINT(1 2)', 2), '*FF*FF212');
st_relate
-----
t
```

Testing a custom spatial relationship pattern as a query condition, with && to enable using a spatial index.

```
-- Find compounds that properly intersect (not just touch) a poly (Interior Intersects)

SELECT c.* , p.name As poly_name
  FROM polys AS p
 INNER JOIN compounds As c
    ON c.geom && p.geom
    AND ST_Relate(p.geom, c.geom, 'T*****');
```

Computing the intersection matrix for spatial relationships.

```
SELECT ST_Relate( 'POINT(1 2)',
                  ST_Buffer( 'POINT(1 2)', 2));
-----
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
```

☒☒

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);

☒☒

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.

GEOS ☒☒☒☒☒

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

```
SELECT ST_RelateMatch('101202FFF', 'TTTTTTFFF') ;
-- result --
t
```

Patterns for common spatial relationships matched against intersection matrix values, for a line in various positions relative to a polygon

```
SELECT pat.name AS relationship, pat.val AS pattern,
       mat.name AS position, mat.val AS matrix,
       ST_RelateMatch(mat.val, pat.val) AS match
FROM (VALUES ( 'Equality', 'T1FF1FFF1' ),
      ( 'Overlaps', 'T*T***T**' ),
      ( 'Within', 'T*F**F***' ),
      ( 'Disjoint', 'FF*FF****' )) AS pat(name,val)
CROSS JOIN
  (VALUES ('non-intersecting', 'FF1FF0212'),
    ('overlapping', '1010F0212'),
    ('inside', '1FF0FF212')) AS mat(name,val);
```

| relationship | pattern   | position         | matrix    | match |
|--------------|-----------|------------------|-----------|-------|
| Equality     | T1FF1FFF1 | non-intersecting | FF1FF0212 | f     |
| Equality     | T1FF1FFF1 | overlapping      | 1010F0212 | f     |
| Equality     | T1FF1FFF1 | inside           | 1FF0FF212 | f     |
| Overlaps     | T*T***T** | non-intersecting | FF1FF0212 | f     |
| Overlaps     | T*T***T** | overlapping      | 1010F0212 | t     |
| Overlaps     | T*T***T** | inside           | 1FF0FF212 | f     |
| Within       | T*F**F*** | non-intersecting | FF1FF0212 | f     |
| Within       | T*F**F*** | overlapping      | 1010F0212 | f     |
| Within       | T*F**F*** | inside           | 1FF0FF212 | t     |
| Disjoint     | FF*FF**** | non-intersecting | FF1FF0212 | t     |
| Disjoint     | FF*FF**** | overlapping      | 1010F0212 | f     |
| Disjoint     | FF*FF**** | inside           | 1FF0FF212 | f     |

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);

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:  $ST\_Touches(A, B) \Leftrightarrow (Int(A) \cap Int(B) = \emptyset) \wedge (A \cap B \neq \emptyset)$

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.

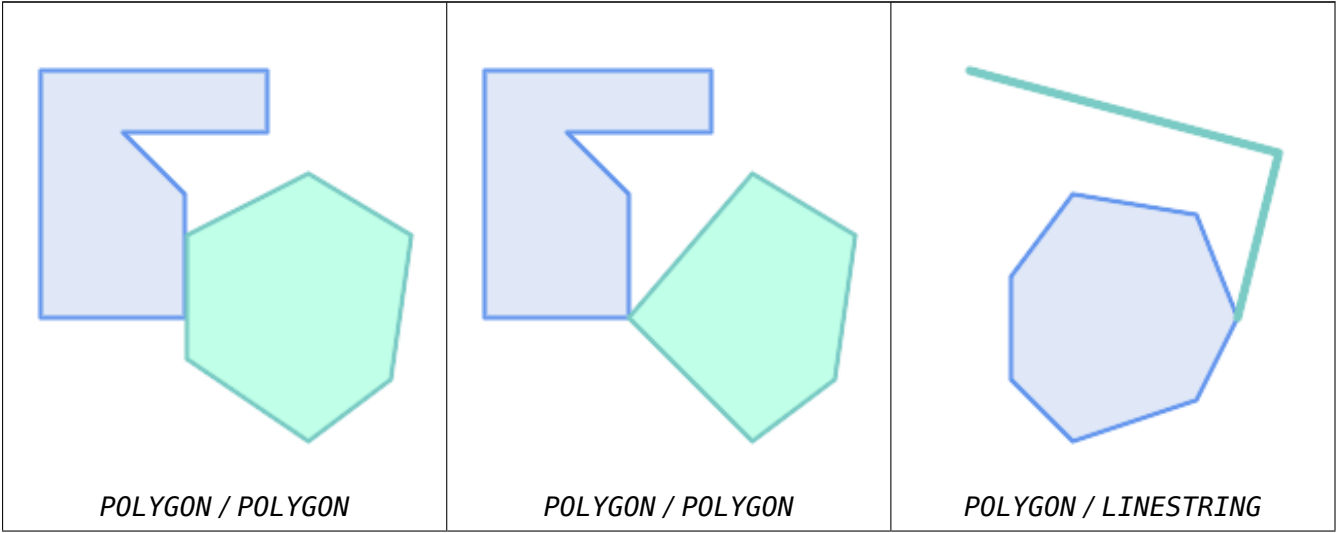


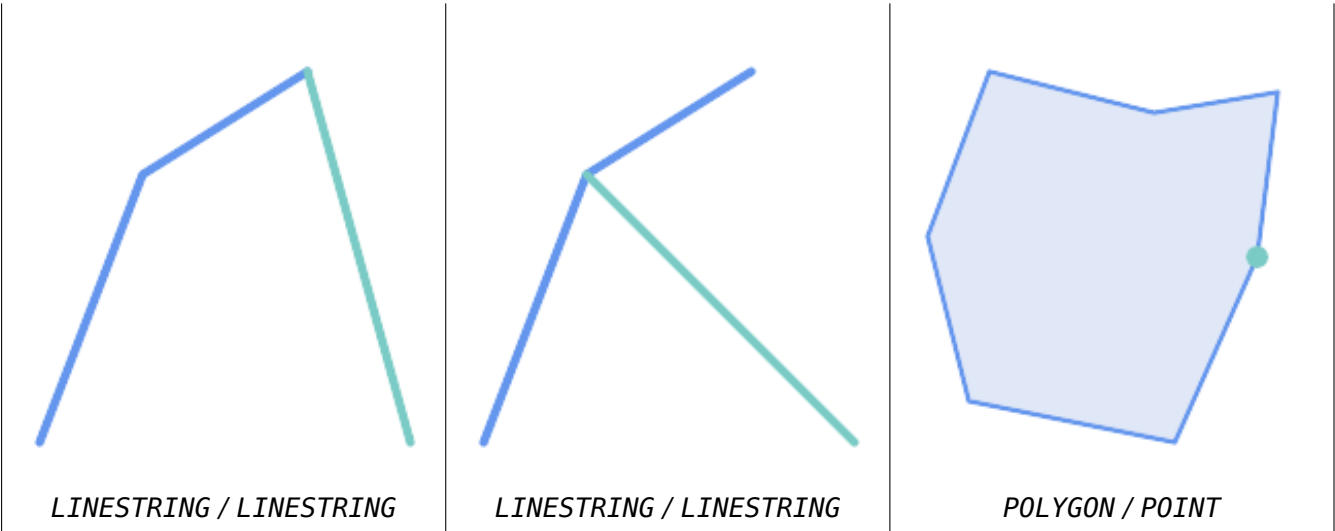
**Important**  
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

- ✓ 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



The `ST_Touches` predicate returns `TRUE` in the following examples.





```
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(1 1)::geometry');
st_touches
-----
f
(1 row)

SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(0 2)::geometry');
st_touches
-----
t
(1 row)
```

7.11.1.16 ST\_Within

ST\_Within — Tests if every point of A lies in B, and their interiors have a point in common

Synopsis

boolean **ST\_Within**(geometry A, geometry B);



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:  $ST\_Within(A, B) \Leftrightarrow (A \sqsubseteq B = A) \wedge (Int(A) \sqcap Int(B) \neq \emptyset)$

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`.

GEOS

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\*\*\*')



This method implements the SQL/MM specification. SQL-MM 3: 5.1.30

```
--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)
```



☒☒

[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);

☒☒

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 ?

2.0.0 ☒☒☒☒☒☒☒☒☒☒.

☒☒

```
-- 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
```

☒☒

[ST\\_3DDFullyWithin](#), [ST\\_DWithin](#), [ST\\_DFullyWithin](#), [ST\\_3DDistance](#), [ST\\_Distance](#), [ST\\_3DMaxDistance](#), [ST\\_Transform](#)

### 7.11.2.2 ST\_3DDFullyWithin

**ST\_3DDFullyWithin** — Tests if two 3D geometries are entirely within a given 3D distance

#### Synopsis

boolean **ST\_3DDFullyWithin**(geometry g1, geometry g2, double precision distance);

☒☒

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.

2.0.0 ☒☒☒☒☒☒☒☒☒☒.



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



This function supports Polyhedral surfaces.



☒☒

```
-- 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
```

☒☒

**ST\_3DDWithin, ST\_DWithin, ST\_DFullyWithin, ST\_3DMaxDistance**

### 7.11.2.3 ST\_DFullyWithin

ST\_DFullyWithin — Tests if a geometry is entirely inside a distance of another

#### Synopsis

boolean **ST\_DFullyWithin**(geometry g1, geometry g2, double precision distance);

☒☒

Returns true if g2 is entirely within distance of g1. Visually, the condition is true if g2 is contained within a distance buffer of g1. The distance is specified in units defined by the spatial reference system of the geometries.



#### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

1.5.0 ☒☒☒☒☒☒☒☒☒☒.

Changed: 3.5.0 : the logic behind the function now uses a test of containment within a buffer, rather than the ST\_MaxDistance algorithm. Results will differ from prior versions, but should be closer to user expectations.

☒☒

```
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 (VALUES
  ('POINT(1 1)', 'LINESTRING(1 5, 2 7, 1 9, 14 12)')
) AS v(geom_a, geom_b)
```

|                   |  |           |  |                |
|-------------------|--|-----------|--|----------------|
| dfullywithin10    |  | dwithin10 |  | dfullywithin20 |
| -----+-----+----- |  |           |  |                |
| f                 |  | t         |  | t              |

☒☒

[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);
```

☒☒

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.

☒☒

```
-- 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);
```

☒☒

**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);

☒☒

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

SQL

```
SELECT ST_PointInsideCircle(ST_Point(1,2), 0.5, 2, 3);
       st_pointinsidecircle
-----
t
```

SQL

**ST\_DWithin**

## 7.12 Measurement Functions

### 7.12.1 ST\_Area

ST\_Area — Returns the area of a geometry.

#### Synopsis

```
float ST_Area(geometry g1);
float ST_Area(geography geog, boolean use_spheroid = true);
```

SQL

**ST\_Area(geometry g1)** - ST\_Surface or ST\_MultiSurface geometries - returns the area of the geometry in square units. SRID 0 returns the area in square units. SRID 4326 returns the area in square meters. ST\_Area(geog,false) returns the area in square meters.

**ST\_Area(geography geog, boolean use\_spheroid = true)** - Returns the area of a geography in square meters. (polyhedral surface) returns the area.

**ST\_Area(geography geog, boolean use\_spheroid = true)** - Returns the area of a geography in square meters. GeographicLib is used for the calculation. Proj 4.9.0 is used for the calculation.

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

XXXXXXXXXXXX, (2.5 XXXXXXXX) 2 XXXXXXXXXXXXXXXXXXXX. 2.5 XXXXXXX 0 XXXX (non-zero) XXXXXXXXXXXXXXXX, XY XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.



 (plot) . EPSG:2249 .

[illegible][illegible]

```

select ST_Area(geom) sqft,
       ST_Area(ST_Transform(geom, 26986)) As sqm
from (
    select
        'SRID=2249;POLYGON((743238 2967416,743238 2967450,
        743265 2967450,743265.625 2967416,743238 2967416))' :: geometry geom
    ) subquery;

```

本圖係根據 1984 年 1 月 1 日之 WGS84 4326 座標系統繪製。圖中所有之經緯度均係根據 WGS84 4326 座標系統計算。圖中所有之經緯度均係根據 WGS84 4326 座標系統計算。

```
select ST_Area(geog) / 0.3048 ^ 2 sqft_spheroid,
       ST_Area(geog, false) / 0.3048 ^ 2 sqft_sphere,
       ST_Area(geog) sqm_spheroid
from (
```

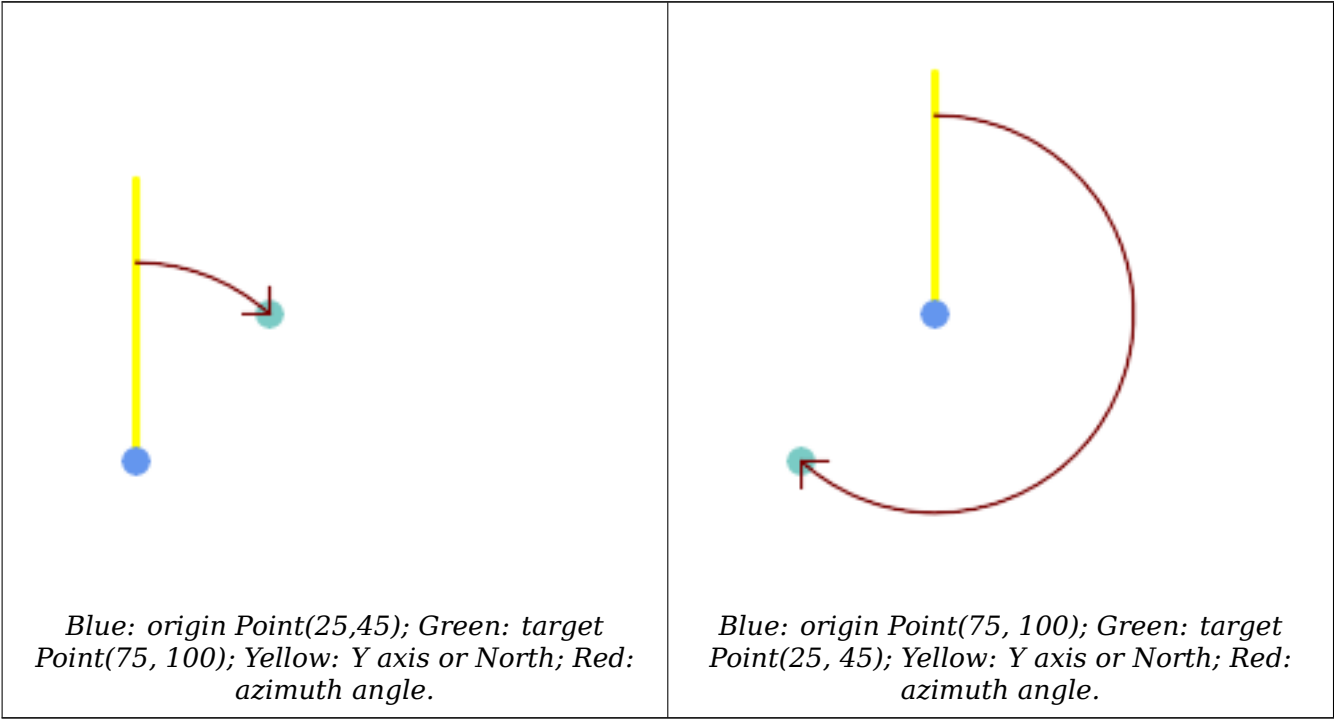
The azimuth is a mathematical concept defined as the angle between a reference vector and a point, with angular units in radians. The result value in radians can be converted to degrees using the PostgreSQL function `degrees()`.

ST\_Translate . [Plpgsqlfunctions PostGIS wiki section](#) upgis\_lineshift .

1.1.0 .  
2.0.0 .  
2.2.0 GeographicLib .  
Proj 4.9.0 .

```
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 |



[ST\\_Angle](#), [ST\\_Translate](#), [ST\\_Project](#), [PostgreSQL Math Functions](#)

### 7.12.3 ST\_Angle

ST\_Angle — 3 (longest) .

## Synopsis

float **ST\_Angle**(geometry point1, geometry point2, geometry point3, geometry point4);  
 float **ST\_Angle**(geometry line1, geometry line2);

返回

返回 3 个值 (longest) 的度数。

**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

PostgreSQL 的 `degrees()` 函数返回度数。返回的度数范围是 0 到 360 度。

Note that `ST_Angle(P1,P2,P3) = ST_Angle(P2,P1,P2,P3)`.

Availability: 2.5.0

返回

返回 3 个值 (longest) 的度数。

```
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
-----
269.999999999999
```

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
```

返回

**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

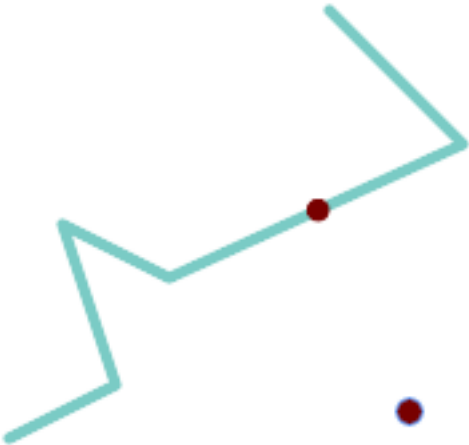
geometry **ST\_ClosestPoint**(geometry geom1, geometry geom2);  
geography **ST\_ClosestPoint**(geography geom1, geography geom2, boolean use\_spheroid = true);

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**  
3 **ST\_3DClosestPoint**

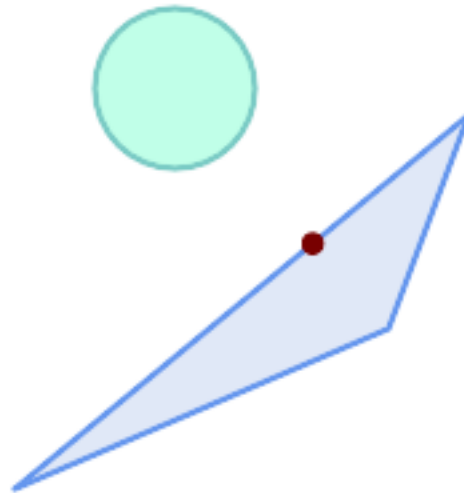
Enhanced: 3.4.0 - Support for geography.  
1.5.0



*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)
```



ST 3DClosestPoint, ST Distance, ST LongestLine, ST ShortestLine, ST MaxDistance

### 7.12.5 ST\_3DClosestPoint

ST\_3DClosestPoint — g2  g1  3                   3D                                                                                                                     

## Synopsis

```
geometry ST_3DClosestPoint(geometry g1, geometry g2);
```



$\text{g}_2 \otimes \dots \otimes \text{g}_1 \otimes \dots \otimes 3 \otimes \dots \otimes \dots$ .  $\dots \otimes 3D \otimes \dots \otimes \dots$ .  $3D \otimes \dots \otimes 3D \otimes \dots \otimes \dots$ .

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

 This function supports Polyhedral surfaces.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

Figure 2.2.0: 2D plot of  $\rho(x, y, z)$ , ( $\rho(x, y, z) \geq 0$  for all  $x, y, z$ ) 2D plot of  $\rho(x, y, z)$ . 2D plot of  $\rho(x, y, z)$ , ( $\rho(x, y, z) \geq 0$  for all  $x, y, z$ ).



|                                                                                                                                                                                                                                                                                                                                                                                         |              |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| ST_ClosestPoint -- 3D, 2D 最近點 (closest point)                                                                                                                                                                                                                                                                                                                                           |              |
| <pre>SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,        ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt FROM (SELECT 'POINT(100 100 30)::geometry As pt,             'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)::'::        geometry As line       ) As foo;</pre>                                                                                   |              |
| cp3d_line_pt                                                                                                                                                                                                                                                                                                                                                                            | ↔            |
| cp2d_line_pt                                                                                                                                                                                                                                                                                                                                                                            |              |
| -----+                                                                                                                                                                                                                                                                                                                                                                                  |              |
| POINT(54.6993798867619 128.935022917228 11.5475869506606)   POINT(73.0769230769231 115.384615384615)                                                                                                                                                                                                                                                                                    |              |
| ST_ClosestPoint -- 3D, 2D 最近點 (closest point)                                                                                                                                                                                                                                                                                                                                           |              |
| <pre>SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,        ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)::geometry As pt,             'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)::'::        geometry As line       ) As foo;</pre>                                                                   |              |
| cp3d_line_pt                                                                                                                                                                                                                                                                                                                                                                            | cp2d_line_pt |
| -----+                                                                                                                                                                                                                                                                                                                                                                                  |              |
| POINT(54.6993798867619 128.935022917228 11.5475869506606)   POINT(50 75)                                                                                                                                                                                                                                                                                                                |              |
| ST_ClosestPoint -- 3D, 2D 最近點 (closest point)                                                                                                                                                                                                                                                                                                                                           |              |
| <pre>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;</pre> |              |
| cp3d                                                                                                                                                                                                                                                                                                                                                                                    | cp2d         |
| -----+                                                                                                                                                                                                                                                                                                                                                                                  |              |
| POINT(39.993580415989 54.1889925532825 5)   POINT(20 40)                                                                                                                                                                                                                                                                                                                                |              |

ST

ST\_AsEWKT, ST\_ClosestPoint, ST\_3DDistance, ST\_3DShortestLine

### 7.12.6 ST\_Distance

ST\_Distance — 3D 最長 (longest) 最長。

## Synopsis

```
float ST_Distance(geometry g1, geometry g2);  
float ST_Distance(geography geog1, geography geog2, boolean use_spheroid = true);
```



□□□□□□, □□□□□□ 3 □□□□□□□□□□□□□□□□□□□□□□ (SRS □□) □□□□□□.

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.

**1.5.0**

Enhanced: 2.1.0 improved speed for geography. See [Making Geography faster](#) for details.

□□□□: 2.1.0 □□□□□□□□□□□□□□□□□□□□.

GeographicLib 2.2.0  
Proj 4.9.0

Changed: 3.0.0 - does not depend on SFCGAL anymore.



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  $\cos(\text{lat})$  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
```

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 |



ST\_3DDistance, ST\_DWithin, ST\_DistanceSphere, ST\_DistanceSpheroid, ST\_MaxDistance, ST\_HausdorffDistance, ST\_FrechetDistance, ST\_Transform

### 7.12.7 ST\_3DDistance

ST 3DDistance —  $\sqrt{(\text{ST}_1 - \text{ST}_2)^2 + (\text{ST}_1 - \text{ST}_3)^2}$  (SRS  $\text{ST}_1$ ) 3  $\sqrt{(\text{ST}_1 - \text{ST}_2)^2 + (\text{ST}_1 - \text{ST}_3)^2}$ .

## Synopsis

```
float ST_3DDistance(geometry g1, geometry g2);
```



□□□□□□, □□□□□□ 3 □□□□□□□□□□□□□□□□ (SRS □□) □□□□□□.



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 2.0.0.
- SQL-MM: 2.2.0, 2D & 3D, Z & 0.
- Changed: 3.0.0 - SFCGAL version removed

SQL

```
-- 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
```

SQL

[ST\\_Distance](#), [ST\\_3DClosestPoint](#), [ST\\_3DDWithin](#), [ST\\_3DMaxDistance](#), [ST\\_3DShortestLine](#), [ST\\_Transform](#)

## 7.12.8 ST\_DistanceSphere

**ST\_DistanceSphere** — Compute the distance between two points on a sphere. PostGIS 1.5.

### Synopsis

float **ST\_DistanceSphere**(geometry geom1lonlatA, geometry geom1lonlatB, float8 radius=6371008);

注意

PostGIS 2 版本使用 `ST_DistanceSphere`。SRID 4326 使用 `ST_DistanceSphere`。PostGIS 1.5 使用 `ST_DistanceSphere`。

1.5 版本使用 `ST_DistanceSphere`。1.5 版本使用 `ST_DistanceSphere`。

注意: 2.2.0 版本使用 `ST_Distance_Sphere`。

注意

```
SELECT round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38)', 4326)) As numeric),2) As dist_meters,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(geom),32611),
ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) As dist_utm11_meters,
round(CAST(ST_Distance(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38)', 4326)) As numeric),5) As dist_degrees,
round(CAST(ST_Distance(ST_Transform(geom,32611),
ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) As min_dist_line_point_meters
FROM
(SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As geom)
as foo;
dist_meters | dist_utm11_meters | dist_degrees | min_dist_line_point_meters
-----+-----+-----+-----
70424.47 | 70438.00 | 0.72900 | 65871.18
```

注意

`ST_Distance`, `ST_DistanceSphere`

## 7.12.9 ST\_DistanceSphere

`ST_DistanceSphere` — 计算两个几何体之间的球面距离。PostGIS 1.5 版本使用 `ST_DistanceSphere`。

### Synopsis

float **ST\_DistanceSphere**(geometry geom1lonlatA, geometry geom1lonlatB, spheroid measurement\_spheroid)

注意

PostGIS 2 版本使用 `ST_DistanceSphere`。PostGIS 1.5 版本使用 `ST_DistanceSphere`。



### Note

PostGIS 2 版本使用 `ST_DistanceSphere`。PostGIS 1.5 版本使用 `ST_DistanceSphere`。

1.5 使用 `ST_DistanceSpheroid` 和 `ST_DistanceSphere`。1.5 使用 `ST_DistanceSpheroid` 和 `ST_DistanceSphere`。

注意: 2.2.0 版本中 `ST_Distance_Spheroid` 已弃用。

SQL

```
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
```

SQL

[ST\\_Distance](#), [ST\\_DistanceSphere](#)

### 7.12.10 ST\_FrechetDistance

`ST_FrechetDistance` — 返回 3 个值 (shortest) 的 Fréchet 距离。

#### Synopsis

`float ST_FrechetDistance(geometry g1, geometry g2, float densifyFrac = -1);`

SQL

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

使用 `ST_FrechetDistance` 时，如果两个几何体都是点，则返回 0。如果两个几何体都是线，则返回最短距离。如果一个是点，另一个是线，则返回点到线的最短距离。



**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.

GEOS 国国国国国

Availability: 2.4.0 - requires GEOS >= 3.7.0

国国

```
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)
```

国国

**ST\_HausdorffDistance**

### 7.12.11 ST\_HausdorffDistance

`ST_HausdorffDistance` — 国国国国国国 3 国国国 (shortest) 国国国国国国国.

#### Synopsis

```
float ST_HausdorffDistance(geometry g1, geometry g2);
float ST_HausdorffDistance(geometry g1, geometry g2, float densifyFrac);
```

国国

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.

1.5.0



*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);
```

几何

**ST\_FrechetDistance**

### 7.12.12 ST\_Length

ST\_Length — 返回几何对象的长度。

#### Synopsis

```
float ST_Length(geometry a_2dlinestring);  
float ST_Length(geography geog, boolean use_spheroid = true);
```

几何

参数: 几何对象, 可以是 ST\_Curve, ST\_MultiCurve 或 2 个维度的任何几何对象。返回 0 或正数。与 **ST\_Perimeter** 类似。对于 geography 类型, 返回 0 或正数。

对于 geography 类型: 计算使用逆测地线计算。长度单位是米。如果 PostGIS 编译时 PROJ 版本 4.8.0 或更高, 球体由 SRID 指定, 否则默认为 WGS84。如果 use\_spheroid = false, 则计算基于球体而不是球面。

与 ST\_Length2D 类似, 但使用球面几何。



#### Warning

注意: 2.0.0 版本中, ST\_Length 对于 geography 类型使用球面几何。2.0.0 版本之前, ST\_Length 对于 geography 类型使用平面几何。2.0.0 版本中, ST\_Length 对于 geography 类型使用球面几何。与 ST\_Perimeter 类似。



#### Note

注意: ST\_Length(gg,false) 返回平面长度。ST\_Length(gg,true) 返回球面长度。



此方法实现了 **OGC Simple Features Implementation Specification for SQL 1.1** s2.1.5.1。



此方法实现了 SQL/MM 规范。SQL-MM 3: 7.1.2, 9.3.4 1.5.0 版本。

几何

对于 EPSG:2249 投影, 返回平面长度。对于其他投影, 返回球面长度。

```

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

```

中文

WGS84 地理坐标系转换为平面坐标系。

```

-- 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

```

中文

**ST\_GeographyFromText, ST\_GeomFromEWKT, ST\_LengthSpheroid, ST\_Perimeter, ST\_Transform**

### 7.12.13 ST\_Length2D

ST\_Length2D — 返回 2D 平面上的长度。返回 ST\_Length 返回的结果。

#### Synopsis

float **ST\_Length2D**(geometry a\_2dlinestring);

中文

返回 2D 平面上的长度。返回 ST\_Length 返回的结果。

文档

[ST\\_Length](#), [ST\\_3DLength](#)

### 7.12.14 ST\_3DLength

ST\_3DLength — 返回 3D 线段的长度。

#### Synopsis

```
float ST_3DLength(geometry a_3dlinestring);
```

文档

返回 3D 线段的长度。2D 线段的长度由 ST\_Length 返回。2D 线段的长度由 ST\_Length2D 返回。ST\_Length 和 ST\_Length2D 返回浮点值。

- ✓ 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
- 版本: 2.0.0 引入 ST\_Length3D 函数。

文档

3D 线段的长度由 ST\_3DLength 返回。EPSG:2249 坐标系下的 3D 线段的长度由 ST\_3DLength 返回。

```
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
```

文档

[ST\\_Length](#), [ST\\_Length2D](#)

### 7.12.15 ST\_LengthSpheroid

ST\_LengthSpheroid — 返回球面线段的长度。

#### Synopsis

```
float ST_LengthSpheroid(geometry a_geometry, spheroid a_spheroid);
```

¶¶

¶¶¶¶¶¶¶¶¶¶/¶¶¶¶¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶/¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶  
¶¶¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶:

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

¶¶¶¶

```
SPHEROID["GRS_1980",6378137,298.257222101]
```

1.2.2 ¶¶¶¶¶¶¶¶¶¶.

¶¶¶¶: 2.2.0 ¶¶¶¶¶¶¶¶ ST\_Length\_Spheroid ¶¶¶¶¶¶¶¶, ST\_3DLength\_Spheroid ¶¶¶¶  
¶¶¶¶¶¶¶¶.



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

¶¶

```
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
```

¶¶

ST\_GeometryN, ST\_Length

## 7.12.16 ST\_LongestLine

ST\_LongestLine — 返回 3 维 (longest) 最长线。

### Synopsis

geometry **ST\_LongestLine**(geometry g1, geometry g2);

返回

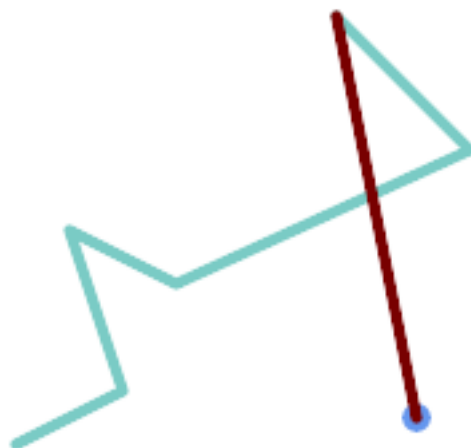
Returns the 2-dimensional longest line between the points of two geometries. The line returned starts on g1 and ends on g2.

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](#).

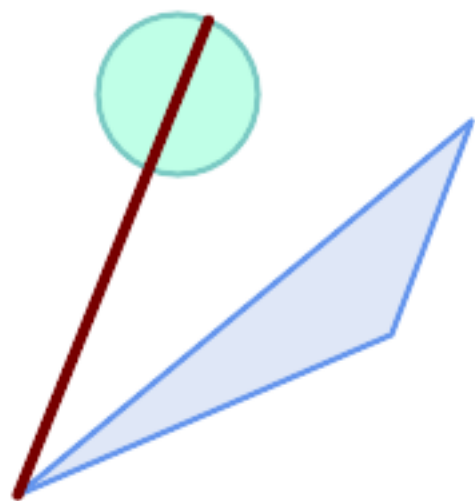
1.5.0 首次引入。

返回



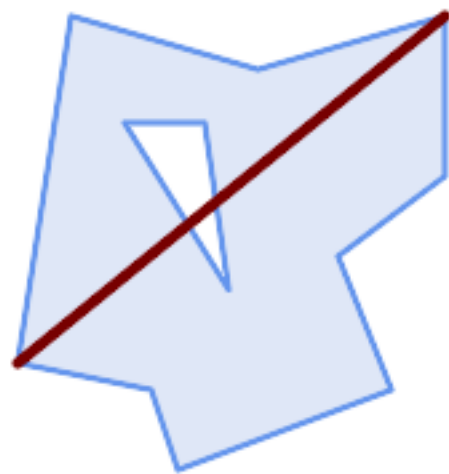
SQL 示例

```
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)
```



🌐🌐🌐🌐🌐🌐🌐🌐🌐🌐🌐

```
SELECT ST_AsText( ST_LongestLine(
    'POLYGON ((190 150, 20 10, 160 70, 190 150))',
    ST_Buffer('POINT(80 160)', 30)
) ) AS llinewkt;
-----
LINESTRING(20 10,105.3073372946034 186.95518130045156)
```



*Longest line across a single geometry. The length of the line is equal to the Maximum Distance. The endpoints of the line lie on the Minimum Bounding Circle.*

```
SELECT ST_AsText( ST_LongestLine( geom, geom)) AS llinewkt,
       ST_MaxDistance( geom, geom) AS max_dist,
       ST_Length( ST_LongestLine(geom, geom)) AS lenll
FROM (SELECT 'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 50, 40 180),
       (60 140, 99 77.5, 90 140, 60 140))'::geometry AS geom) AS t;

  llinewkt      |      max_dist      |      lenll
-----+-----+-----
LINESTRING(20 50,180 180) | 206.15528128088303 | 206.15528128088303
```





ST\_MaxDistance, ST\_ShortestLine, ST\_3DLongestLine, ST\_MinimumBoundingCircle

### 7.12.17 ST\_3DLongestLine

ST\_3DLongestLine — 3D longest line.

## Synopsis

```
geometry ST_3DLongestLine(geometry g1, geometry g2);
```



3 (longest) . ,   
 g1 g2 . 3   
 ST\_3DMaxDistance g1 g2 .

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

OpenGL: 2.2.0 OpenGL 2D Renderer, (OpenGL Z 0 Renderer) 2D  
 Renderer. 2D to 3D, Z 0 Renderer.

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



⊠⊠⊠⊠⊠⊠⊠⊠ -- 3D, 2D ⊠⊠⊠⊠⊠⊠⊠

```
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) |

3D, 2D 示例

```
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) |

3D, 2D 示例

```
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)
```

3D

[ST\\_3DClosestPoint](#), [ST\\_3DDistance](#), [ST\\_LongestLine](#), [ST\\_3DShortestLine](#), [ST\\_3DMaxDistance](#)

## 7.12.18 ST\_MaxDistance

`ST_MaxDistance` — 返回 2 个几何对象之间的最大距离。

### Synopsis

`float ST_MaxDistance(geometry g1, geometry g2);`

3D

返回 2 个几何对象之间的最大距离。g1 与 g2 可以是点、线、面。

返回 2 个几何对象之间的最大距离。g1 与 g2 可以是点、线、面。

1.5.0 引入。

3D

3D 示例

```
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
```



ST Distance, ST LongestLine, ST DFullyWithin

### 7.12.19 ST\_3DMaxDistance

ST\_3DMaxDistance — ,  (SRS ) 3                                                                                                                                  

## Synopsis

```
float ST_3DMaxDistance(geometry g1, geometry g2);
```



□□□□□□□□, □□□□□□□ 3 □□□□□□□□□□□□□□□□□□□□□□□□□□ (SRS □□) □□□□□□□.

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

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

OpenGL: 2.2.0, 2D, 3D, Z, Z, 0.



```
-- 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

```



ST Distance, ST 3DDWithin, ST 3DMaxDistance, ST Transform

### 7.12.20 ST\_MinimumClearance

ST\_MinimumClearance — □□□□□□ (robustness) □□□□□□□□□□ (clearance) □□□□□□.

## Synopsis

```
float ST_MinimumClearance(geometry g);
```

[illegible]

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.

☐ e ☐, ☐

- 2 e
- e

ST Minimum Clearance 0.0625".

To avoid validity issues caused by precision loss, **ST\_ReducePrecision** can reduce coordinate precision while ensuring that polygonal geometry remains valid.

2.3.0 ☒☒☒☒☒☒☒☒☒☒☒.



```
SELECT ST_MinimumClearance('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))');
st_minimumclearance
-----
0.00032
```



specified by the SRID, otherwise it is exclusive to WGS84. If `use_spheroid = false`, then calculations will approximate a sphere instead of a spheroid.

ST\_Perimeter2D 計算多邊形、多線段、點、圓、圓弧、橢圓、橢圓弧的周長。

✔ 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

版本: 2.0.0 計算周長。

用法:

計算多邊形的周長。SRID 2249 是 EPSG:2249 的 SRID。

```
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)
```

用法:

計算多邊形的周長。SRID 4326 是 WGS84 的 SRID。

```
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
```

[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);

Perimeter of a polygonal geometry in 2D space.



**Note**

Perimeter of a polygonal geometry in 2D space. Alias for `ST_Perimeter`.

[ST\\_Perimeter](#)

### 7.12.24 ST\_3DPerimeter

`ST_3DPerimeter` — Perimeter of a 3D geometry.

#### Synopsis

float **ST\_3DPerimeter**(geometry geomA);

文档

ST\_3DPerimeter 3 维几何体的周长。2 维几何体的周长由 ST\_Perimeter 函数返回。



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

版本: 2.0.0 函数名: ST\_Perimeter3D 返回类型: float。

文档

ST\_3DPerimeter 函数示例

```
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 |

文档

[ST\\_GeomFromEWKT](#), [ST\\_Perimeter](#), [ST\\_Perimeter2D](#)

## 7.12.25 ST\_ShortestLine

ST\_ShortestLine — 返回两个几何体之间的最短 2 维线。

### Synopsis

geometry **ST\_ShortestLine**(geometry geom1, geometry geom2);  
 geography **ST\_ShortestLine**(geography geom1, geography geom2, boolean use\_spheroid = true);

文档

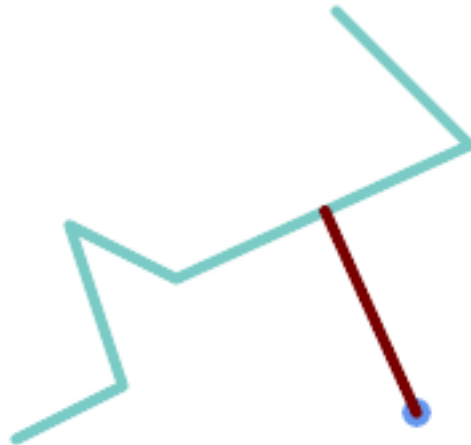
Returns the 2-dimensional shortest line between two geometries. The line returned starts in **geom1** and ends in **geom2**. If **geom1** and **geom2** intersect the result is a line with start and end at an intersection point. The length of the line is the same as [ST\\_Distance](#) returns for g1 and g2.

Enhanced: 3.4.0 - support for geography.

1.5.0 函数名: ST\_ShortestLine。

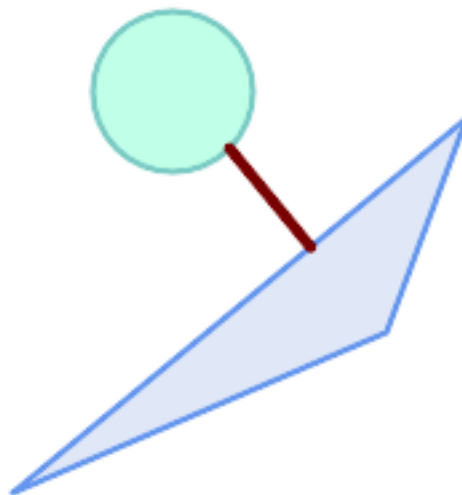


☒☒



*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)
```



ST\_ClosestPoint, ST\_Distance, ST\_LongestLine, ST\_MaxDistance

### 7.12.26 ST\_3DShortestLine

ST\_3DShortestLine — 3D shortest line between two geometries.

## Synopsis

```
geometry ST_3DShortestLine(geometry g1, geometry g2);
```



3 (shortest) . ,  
 . g1 g2 ,  
 . g1 g2 ,  
 . g1 g2 . ST\_3DDistance g1 g2  
 3 ST\_3DDistance g1 g2 .

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

**Figure 2.2.0:** 2D  $\text{Z}$ -axis projection, ( $\text{Z} \geq 0$ ) 2D  $\text{X}$ - $\text{Y}$  plane. 2D  $\rightarrow$  3D  $\text{X}$ ,  $\text{Y}$ ,  $\text{Z}$  axis.

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



XXXXXXXXXX -- 3D, 2D XXXXXXXXX

```
SELECT ST_AsEWKT(ST_3DShortestLine(line,pt)) AS shl3d_line_pt,
        ST_AsEWKT(ST_ShortestLine(line,pt)) As shl2d_line_pt
FROM (SELECT 'POINT(100 100 30) '::geometry As pt,
        'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000) '::
geometry As line
      ) As foo;
```

shl3d\_line\_pt                      shl2d\_line\_pt                      |   ←

```
LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30) | ↩
LINESTRING(73.0769230769231 115.384615384615,100 100)
```

3D, 2D ShortestLine

```
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;

shl2d_line_pt          shl3d_line_pt          |
-----+-----
LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30) | LINESTRING(
(50 75,50 74)
```

3D, 2D ShortestLine

```
SELECT ST_AsEWKT(ST_3DShortestLine(poly, mline)) As shl3d,
       ST_AsEWKT(ST_ShortestLine(poly, mline)) As shl2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,
100 100 5, 175 150 5))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125
100 1, 175 155 1),
       (1 10 2, 5 20 1))') As mline ) As foo;

shl3d          |          shl2d
-----+-----
LINESTRING(39.993580415989 54.1889925532825 5,40.4078575708294 53.6052383805529 5.03423778139177) | LINESTRING(20 40,20 40)
```

ST\_3D

[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);

ST\_ClipByBox2D

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).

GEOS

2.2.0

☒☒

```
-- Rely on implicit cast from geometry to box2d for the second parameter
SELECT ST_ClipByBox2D(geom, ST_MakeEnvelope(0,0,10,10)) FROM mytab;
```

☒☒

[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);

☒☒

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** parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see [ST\\_ReducePrecision](#).

GEOS ☒☒☒☒☒

Enhanced: 3.1.0 accept a **gridSize** parameter.

Requires GEOS  $\geq 3.9.0$  to use the **gridSize** parameter.



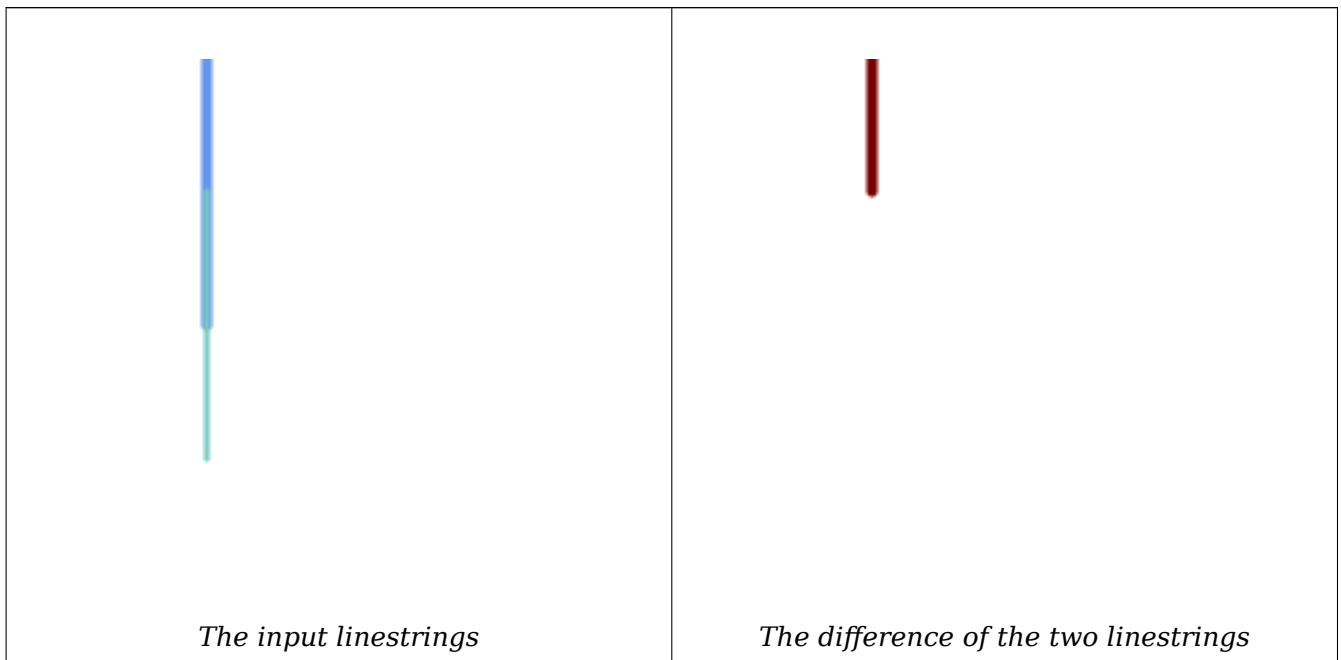
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3](#)



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.



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)
```



**[ST\\_SymDifference](#), [ST\\_Intersection](#), [ST\\_Union](#), [ST\\_ReducePrecision](#)**

### 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 );
```

**Notes**

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` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see [ST\\_ReducePrecision](#).

`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 SFCGAL 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.

GEOS

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.

☒☒

```
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)
```

☒☒

[ST\\_3DIntersection](#), [ST\\_Difference](#), [ST\\_Union](#), [ST\\_ClipByBox2D](#), [ST\\_Dimension](#), [ST\\_Dump](#), [ST\\_Force2D](#), [ST\\_SymDifference](#), [ST\\_Intersects](#), [ST\\_Multi](#), [ST\\_ReducePrecision](#)

### 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);

☒☒

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.

☒☒

```
SELECT id,
       ST_MemUnion(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

☒☒

[ST\\_Union](#)

### 7.13.5 ST\_Node

`ST_Node` — Nodes a collection of lines.

#### Synopsis

geometry **ST\_Node**(geometry geom);





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.

GEOS

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.



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;
-----
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;
-----
MULTILINESTRING((2 5,2 3),(2 3,2 1,4 1),(4 1,2 3),(4 1,6 1),(6 1,7 1))
```



[ST\\_UnaryUnion](#), [ST\\_AsBinary](#)

### 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);



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.

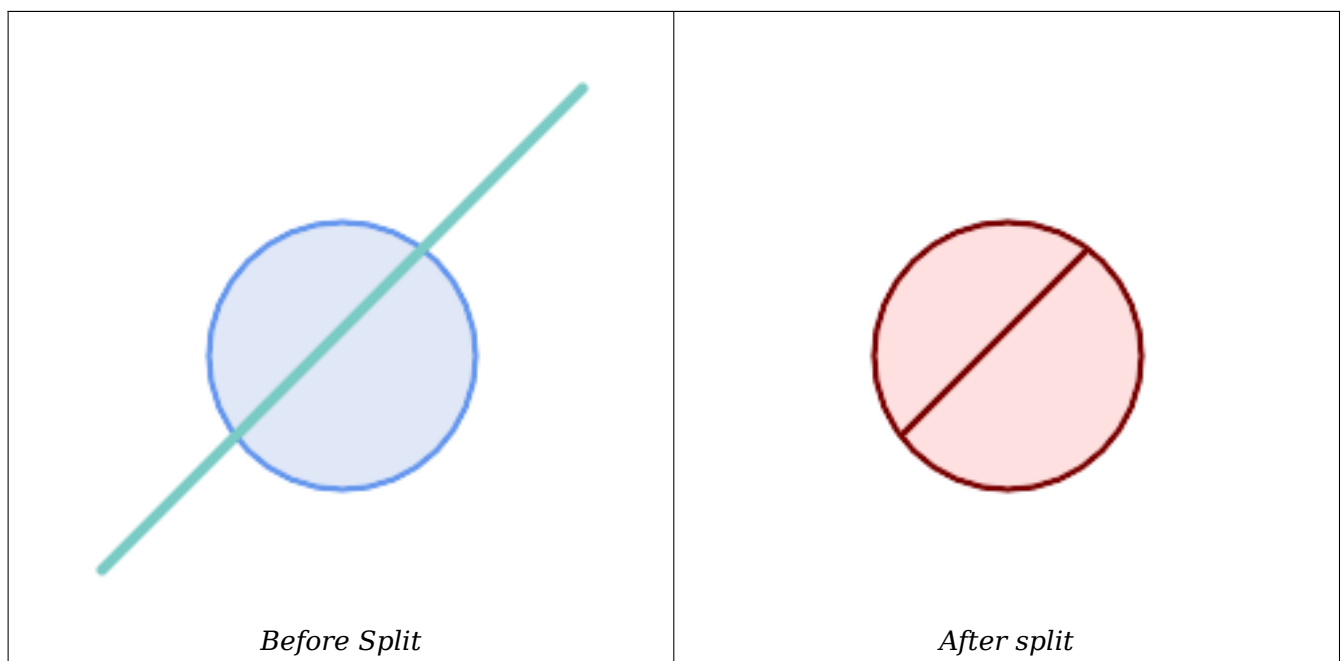
Availability: 2.0.0 requires GEOS

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.



Split a Polygon by a Line.

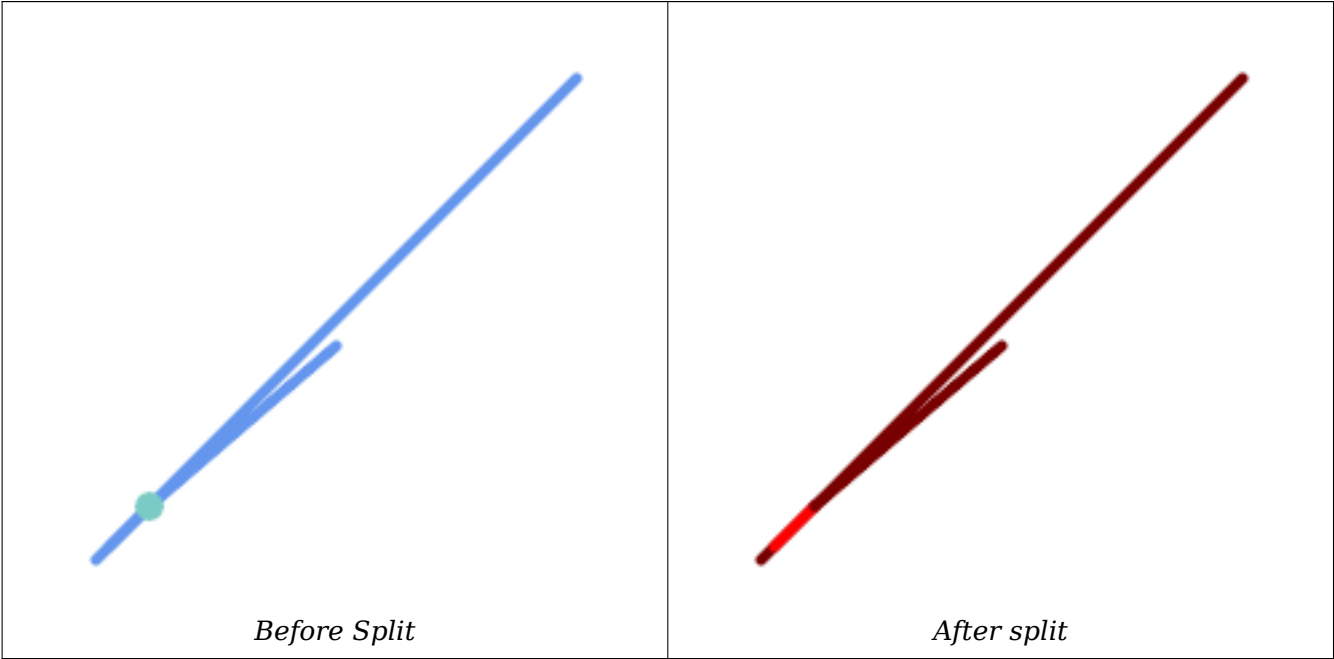


```
SELECT ST_AsText( ST_Split(
    ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50), -- circle
    ST_MakeLine(ST_Point(10, 10),ST_Point(190, 190)) -- line
));

-- result --
GEOMETRYCOLLECTION(
```

```
POLYGON((150 90,149.039264020162 80.2454838991936,146.193976625564  
70.8658283817455,..),  
POLYGON(..))  
)
```

Split a MultiLineString by a Point, where the point lies exactly on both LineStrings elements.



```
SELECT ST_AsText(ST_Split(  
  'MULTILINESTRING((10 10, 190 190), (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(
```



ST\_Snap, ST\_AsBinary

### 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);
```



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.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see [ST ReducePrecision](#).

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.

GEOS ☐☐☐☐☐

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.

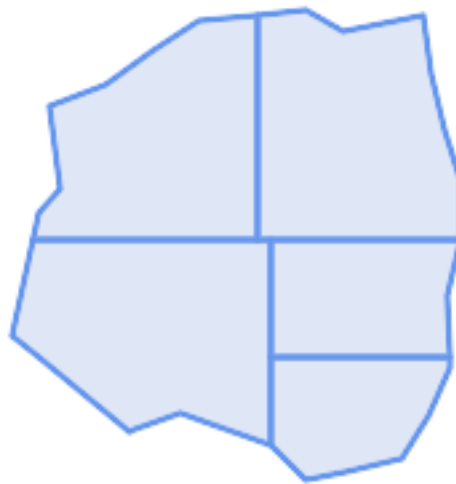
Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5.

Enhanced: 3.1.0 accept a gridSize parameter.

Requires GEOS >= 3.9.0 to use the gridSize parameter



**Example:** Subdivide a polygon into parts with no more than 10 vertices, and assign each part a unique id.

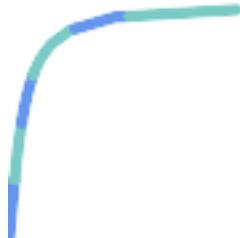


*Subdivided to maximum 10 vertices*

```
SELECT row_number() OVER() As rn, ST_AsText(geom) As wkt
FROM (SELECT ST_SubDivide(
        'POLYGON((132 10,119 23,85 35,68 29,66 28,49 42,32 56,22 64,32 110,40 119,36 150,
        57 158,75 171,92 182,114 184,132 186,146 178,176 184,179 162,184 141,190 122,
        190 100,185 79,186 56,186 52,178 34,168 18,147 13,132 10))'::geometry,10)) AS f(
        geom);
```

| rn | b'' b'' | wkt                                                                                                            |
|----|---------|----------------------------------------------------------------------------------------------------------------|
| 1  | b'' b'' | POLYGON((119 23,85 35,68 29,66 28,32 56,22 64,29.8260869565217 100,119 100,119 23))                            |
| 2  | b'' b'' | POLYGON((132 10,119 23,119 56,186 56,186 52,178 34,168 18,147 13,132 10))                                      |
| 3  | b'' b'' | POLYGON((119 56,119 100,190 100,185 79,186 56,119 56))                                                         |
| 4  | b'' b'' | POLYGON((29.8260869565217 100,32 110,40 119,36 150,57 158,75 171,92 182,114 184,114 100,29.8260869565217 100)) |
| 5  | b'' b'' | POLYGON((114 184,132 186,146 178,176 184,179 162,184 141,190 122,190 100,114 100,114 184))                     |

**Example:** Densify a long geography line using `ST_Segmentize(geography, distance)`, and use `ST_Subdivide` to split the resulting line into sublines of 8 vertices.



*The densified and split lines.*

```
SELECT ST_AsText( ST_Subdivide(
    ST_Segmentize('LINESTRING(0 0, 85 85)::geography',
        1200000)::geometry, 8));
```

```
LINESTRING(0 0,0.487578359029357 5.57659056746196,0.984542144675897 ↵
    11.1527721155093,1.50101059639722 16.7281035483571,1.94532113630331 21.25)
LINESTRING(1.94532113630331 21.25,2.04869538062779 22.3020741387339,2.64204641967673 ↵
    27.8740533545155,3.29994062412787 33.443216802941,4.04836719489742 ↵
    39.0084282520239,4.59890468420694 42.5)
LINESTRING(4.59890468420694 42.5,4.92498503922732 44.5680389206321,5.98737409390639 ↵
    50.1195229244701,7.3290919767674 55.6587646879025,8.79638749938413 60.1969505994924)
LINESTRING(8.79638749938413 60.1969505994924,9.11375579533779 ↵
    61.1785363177625,11.6558166691368 66.6648504160202,15.642041247655 ↵
    72.0867690601745,22.8716627200212 77.3609628116894,24.6991785131552 77.8939011989848)
LINESTRING(24.6991785131552 77.8939011989848,39.4046096622744 ↵
    82.1822848017636,44.7994523421035 82.5156766227011)
LINESTRING(44.7994523421035 82.5156766227011,85 85)
```

**Example:** Subdivide the complex geometries of a table in-place. The original geometry records are deleted from the source table, and new records for each subdivided result geometry are inserted.

```
WITH complex_areas_to_subdivide AS (
    DELETE from polygons_table
    WHERE ST_NPoints(geom)
> 255
    RETURNING id, column1, column2, column3, geom
)
INSERT INTO polygons_table (fid, column1, column2, column3, geom)
    SELECT fid, column1, column2, column3,
        ST_Subdivide(geom, 255) as geom
    FROM complex_areas_to_subdivide;
```

**Example:** Create a new table containing subdivided geometries, retaining the key of the original geometry so that the new table can be joined to the source table. Since `ST_Subdivide` is a set-returning (table) function that returns a set of single-value rows, this syntax automatically produces a table with one row for each result part.

```
CREATE TABLE subdivided_geoms AS
```

```
SELECT pkey, ST_Subdivide(geom) AS geom
FROM original_geoms;
```

☒☒

[ST\\_ClipByBox2D](#), [ST\\_Segmentize](#), [ST\\_Split](#), [ST\\_NPoints](#), [ST\\_ReducePrecision](#)

### 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);

☒☒

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` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see [ST\\_ReducePrecision](#).

GEOS ☒☒☒☒☒

Enhanced: 3.1.0 accept a `gridSize` parameter.

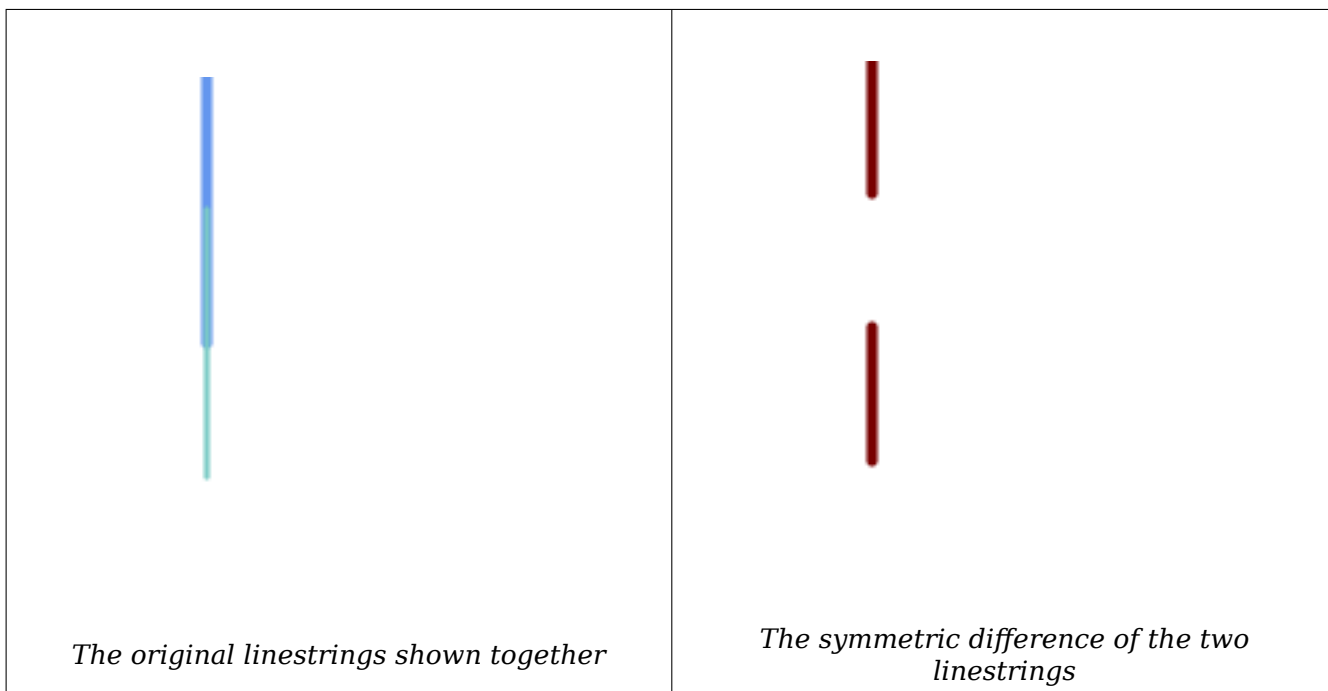
Requires GEOS  $\geq$  3.9.0 to use the `gridSize` parameter

✓ 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.

☒☒



```
--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))
```

☒☒

**[ST\\_Difference](#), [ST\\_Intersection](#), [ST\\_Union](#), [ST\\_ReducePrecision](#)**

### 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);



☒☒

A single-input variant of **ST\_Union**. The input may be a single geometry, a MultiGeometry, or a GeometryCollection. The union is applied to the individual elements of the input.

This function can be used to fix MultiPolygons which are invalid due to overlapping components. However, the input components must each be valid. An invalid input component such as a bow-tie polygon may cause an error. For this reason it may be better to use **ST\_MakeValid**.

Another use of this function is to node and dissolve a collection of linestrings which cross or overlap to make them **simple**. (**ST\_Node** also does this, but it does not provide the `gridSize` option.)

It is possible to combine **ST\_UnaryUnion** with **ST\_Collect** 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` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see **ST\_ReducePrecision**.

✔ 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

2.0.0 ☒☒☒☒☒☒☒☒☒☒.

☒☒

**ST\_Union**, **ST\_MemUnion**, **ST\_MakeValid**, **ST\_Collect**, **ST\_Node**, **ST\_ReducePrecision**

### 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);
```

☒☒

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>

If the optional gridSize parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see [ST\\_ReducePrecision](#).



#### Note

[ST\\_Collect](#) 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.

GEOS

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.

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))
```

☒☒

[ST\\_Collect](#), [ST\\_UnaryUnion](#), [ST\\_MemUnion](#), [ST\\_Intersection](#), [ST\\_Difference](#), [ST\\_SymDifference](#), [ST\\_Reduce](#)

## 7.14 ☒☒☒☒☒☒

### 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);
```

☒☒

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 can handle invalid inputs and the output is always a valid polygonal geometry. Buffering by distance 0 is sometimes used as a way of repairing invalid polygons. **ST\_MakeValid** is more suitable for this process as it can handle multi-polygons.

**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.

GEOS 

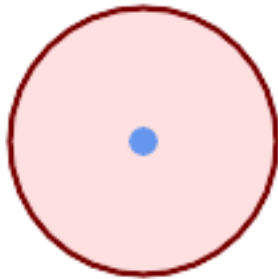


This method implements the **OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3**



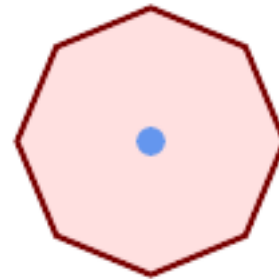
This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.30





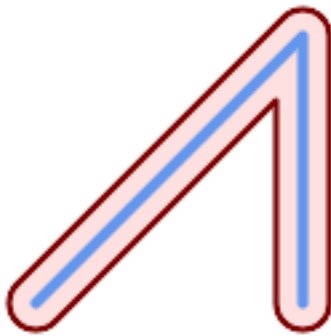
*quad\_segs=8 (国国国)*

```
SELECT ST_Buffer(  
  ST_GeomFromText('POINT(100 90)'),  
  50, 'quad_segs=8');
```



*quad\_segs=2 (国国)*

```
SELECT ST_Buffer(  
  ST_GeomFromText('POINT(100 90)'),  
  50, 'quad_segs=2');
```



*endcap=round join=round (国国国)*

```
SELECT ST_Buffer(  
  ST_GeomFromText(  
    'LINESTRING(50 50,150 150,150 50)'  
  ), 10, 'endcap=round join=round');
```



*endcap=square*

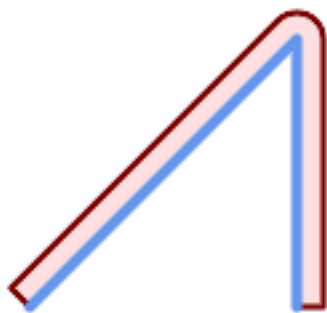
```
SELECT ST_Buffer(  
  ST_GeomFromText(  
    'LINESTRING(50 50,150 150,150 50)'  
  ), 10, 'endcap=square join=round');
```

*join=bevel*

```
SELECT ST_Buffer(  
  ST_GeomFromText(  
    'LINESTRING(50 50,150 150,150 50)'  
  ), 10, 'join=bevel');
```

*join=mitre mitre\_limit=5.0 (国国国国国国国国)*

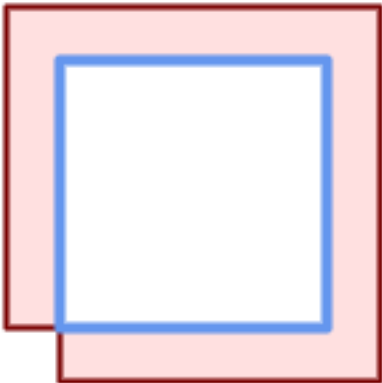
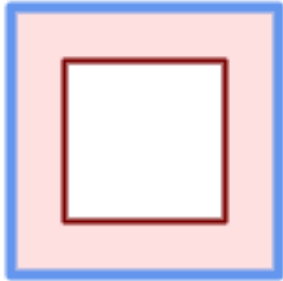
```
SELECT ST_Buffer(  
  ST_GeomFromText(  
    'LINESTRING(50 50,150 150,150 50)'  
  ), 10, 'join=mitre mitre_limit=5.0');
```

*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');
```

|                                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                                                                                               |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  <p><i>right-hand-winding, polygon boundary<br/>side=left</i></p> <pre>SELECT ST_Buffer(<br/>  ST_ForceRHR(<br/>    ST_Boundary(<br/>      ST_GeomFromText(<br/>'POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))'),<br/>    ), 20, 'side=left');</pre> |  <p><i>right-hand-winding, polygon boundary<br/>side=right</i></p> <pre>SELECT ST_Buffer(<br/>  ST_ForceRHR(<br/>    ST_Boundary(<br/>      ST_GeomFromText(<br/>'POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))'),<br/>    ), 20, 'side=right');</pre> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```
--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))
```



[ST\\_Collect](#), [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);

Creates an areal geometry formed by the constituent linework of the input geometry. The input can be a LineString, MultiLineString, Polygon, MultiPolygon or a GeometryCollection. The result is a Polygon or MultiPolygon, depending on input. If the input linework does not form polygons, NULL is returned.

Unlike [ST\\_MakePolygon](#), this function accepts rings formed by multiple lines, and can form any number of polygons.

This function converts inner rings into holes. To turn inner rings into polygons as well, use [ST\\_Polygonize](#).

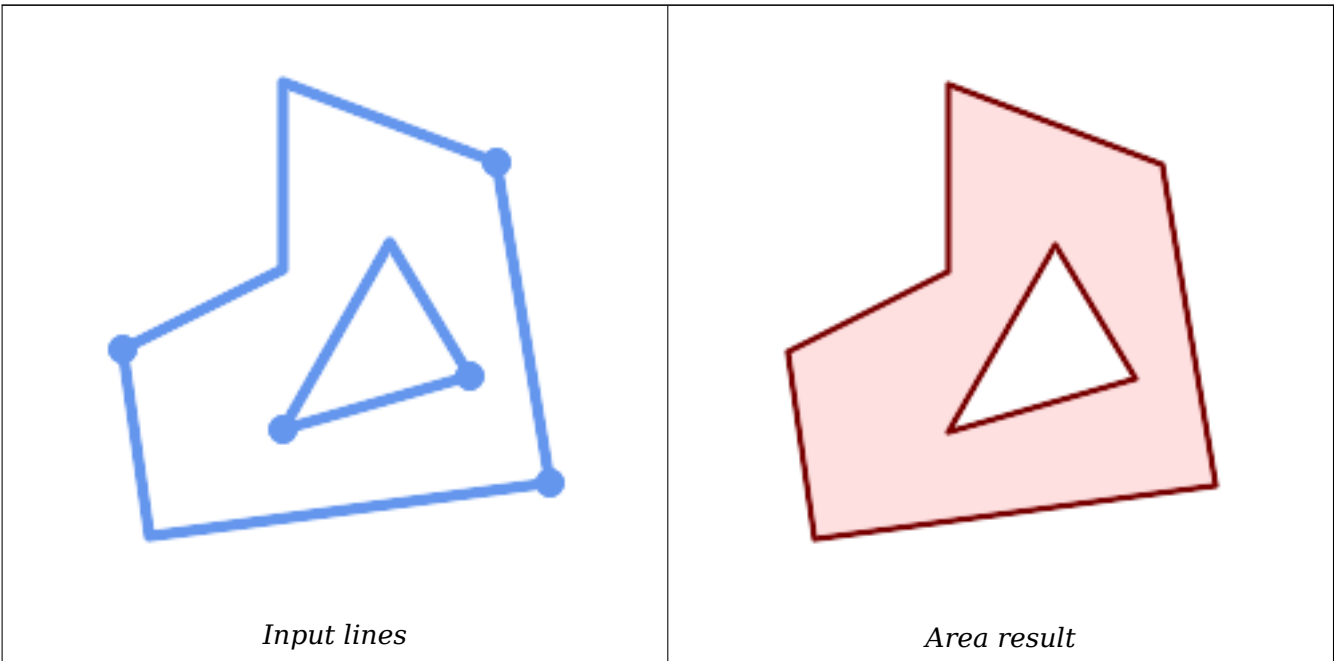


**Note**

Input linework must be correctly noded for this function to work properly. [ST\\_Node](#) can be used to node lines.

If the input linework crosses, this function will produce invalid polygons. [ST\\_MakeValid](#) can be used to ensure the output is valid.

1.1.0



```

WITH data(geom) AS (VALUES
  ('LINESTRING (180 40, 30 20, 20 90)::geometry)
, ('LINESTRING (180 40, 160 160)::geometry)
, ('LINESTRING (160 160, 80 190, 80 120, 20 90)::geometry)
, ('LINESTRING (80 60, 120 130, 150 80)::geometry)
, ('LINESTRING (80 60, 150 80)::geometry)
)
SELECT ST_AsText( ST_BuildArea( ST_Collect( geom )))
FROM data;
-----
POLYGON((180 40,30 20,20 90,80 120,80 190,160 160,180 40),(150 80,120 130,80 60,150 80))

```



*Create a donut from two circular polygons*

```

SELECT ST_BuildArea(ST_Collect(inring,outring))
FROM (SELECT
  ST_Buffer('POINT(100 90)', 25) As inring,
  ST_Buffer('POINT(100 90)', 50) As outring) As t;

```

☒☒

[ST\\_Collect](#), [ST\\_MakePolygon](#), [ST\\_MakeValid](#), [ST\\_Node](#), [ST\\_Polygonize](#), [ST\\_BdPolyFromText](#), [ST\\_BdMPolyFr](#)  
(wrappers to this function with standard OGC interface)

### 7.14.3 ST\_Centroid

ST\_Centroid — ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

#### Synopsis

```

geometry ST_Centroid(geometry g1);
geography ST_Centroid(geography g1, boolean use_spheroid = true);

```



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](#).

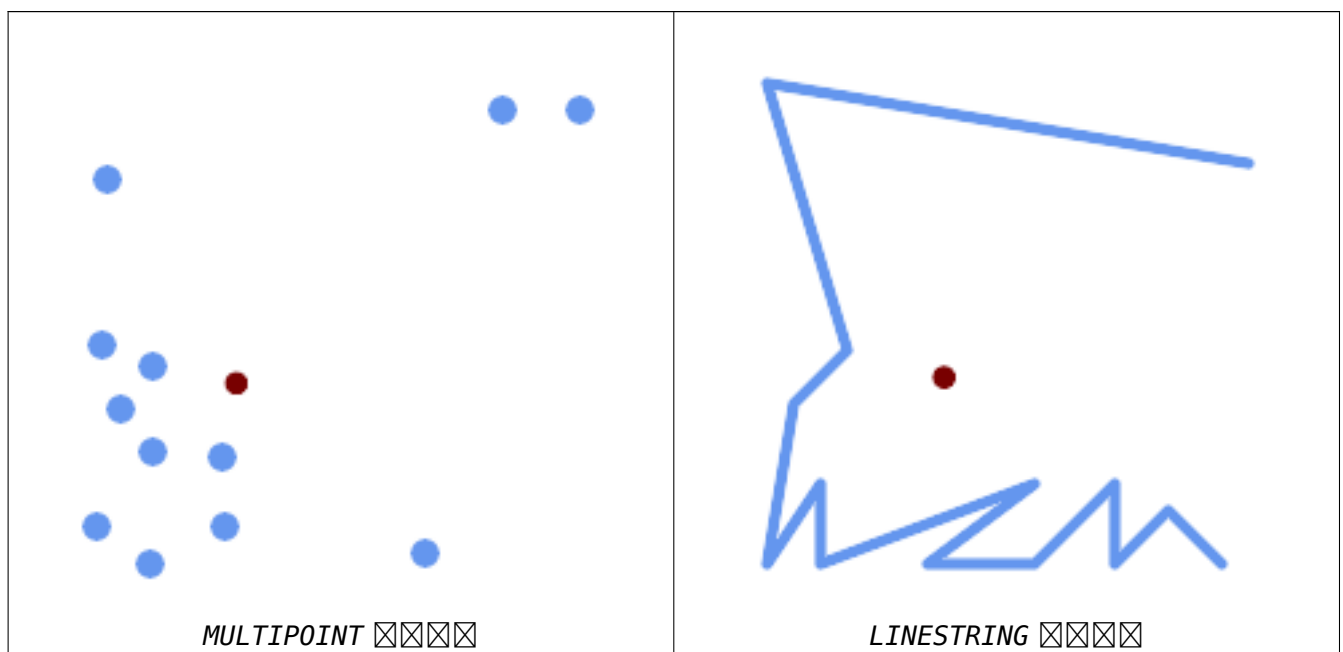
New in 2.3.0 : supports CIRCULARSTRING and COMPOUNDCURVE (using CurveToLine)

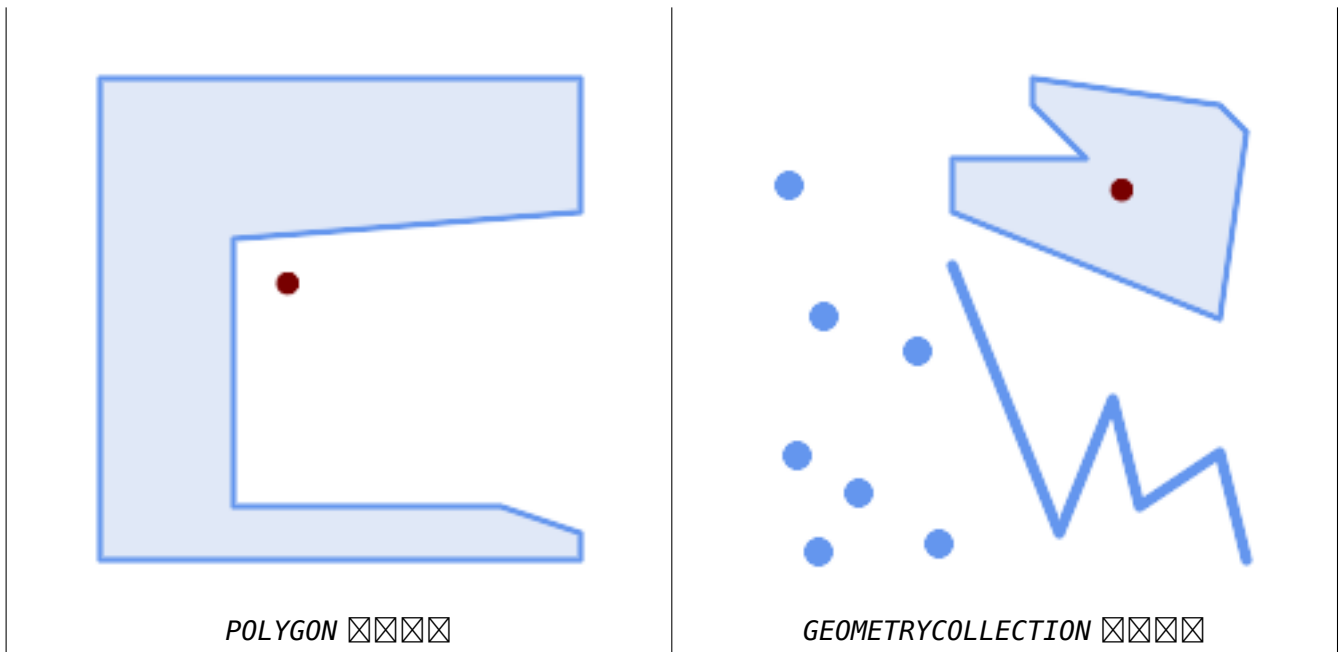
Availability: 2.4.0 support for geography was introduced.

- ✓ 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



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 )'));
```

-----

```
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)
```

☒☒

[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);



Smooths a linear or polygonal geometry using **Chaikin's algorithm**. The degree of smoothing is controlled by the `nIterations` parameter. On each iteration, each interior vertex is replaced by two vertices located at 1/4 of the length of the line segments before and after the vertex. A reasonable degree of smoothing is provided by 3 iterations; the maximum is limited to 5.

If `preserveEndPoints` is true, the endpoints of Polygon rings are not smoothed. The endpoints of LineStrings are always preserved.



### Note

The number of vertices doubles with each iteration, so the result geometry may have many more points than the input. To reduce the number of points use a simplification function on the result (see [ST\\_Simplify](#), [ST\\_SimplifyPreserveTopology](#) and [ST\\_SimplifyVW](#)).

The result has interpolated values for the Z and M dimensions when present.



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

Availability: 2.5.0



### Smoothing a triangle:

```
SELECT ST_AsText(ST_ChaikinSmoothing(geom)) smoothed
FROM (SELECT 'POLYGON((0 0, 8 8, 0 16, 0 0))'::geometry geom) AS foo;
```

smoothed

b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' ' ←

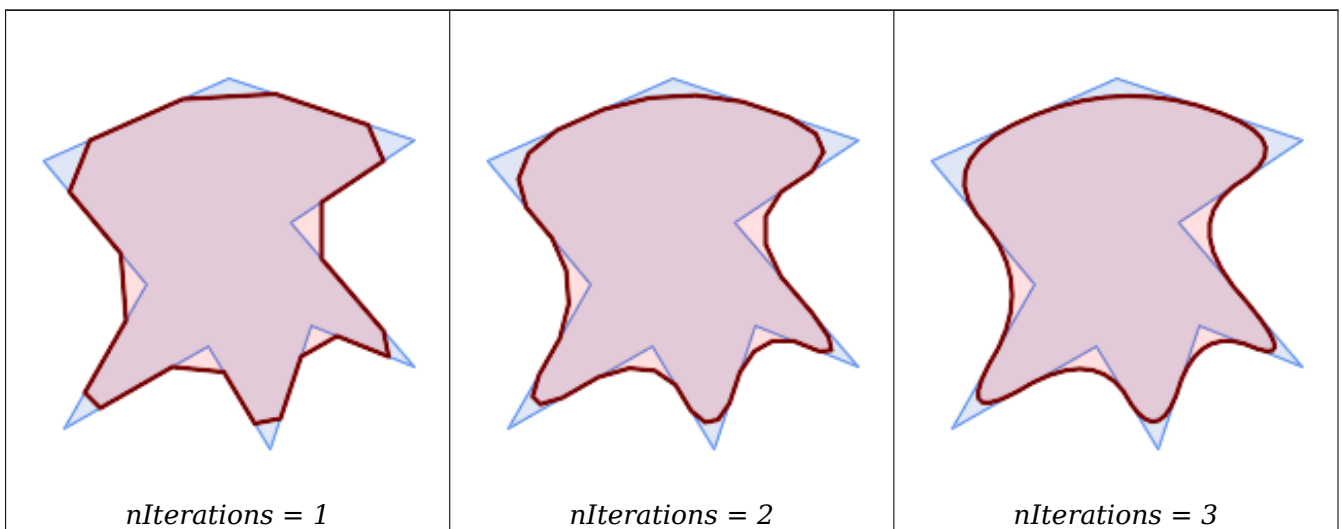
b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' ' ←

'-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' ' ←

'-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' ' ←

POLYGON((2 2,6 6,10,2 14,0 12,0 4,2 2))

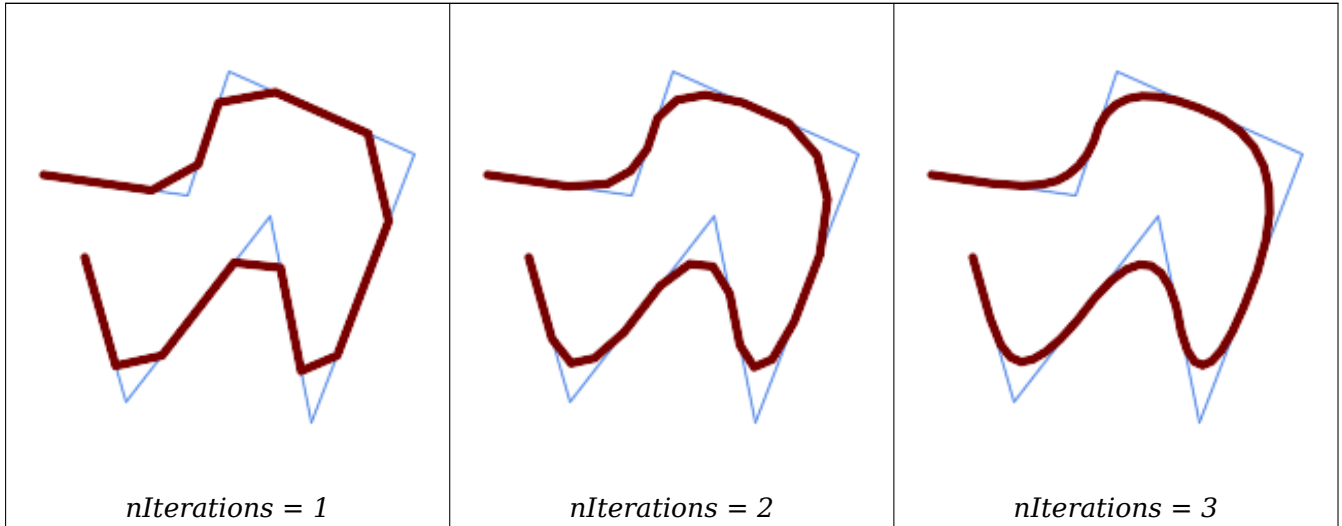
Smoothing a Polygon using 1, 2 and 3 iterations:



```
SELECT ST_ChaikinSmoothing(
```

```
'POLYGON ((20 20, 60 90, 10 150, 100 190, 190 160, 130 120, 190 50, 140 70, 120 10, 90 60, 20 20))',
generate_series(1, 3) );
```

Smoothing a LineString using 1, 2 and 3 iterations:



```
SELECT ST_ChaikinSmoothing(
  'LINESTRING (10 140, 80 130, 100 190, 190 150, 140 20, 120 120, 50 30, 30 100)'
  ,
  generate_series(1, 3) );
```

☒☒

[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);

☒☒

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

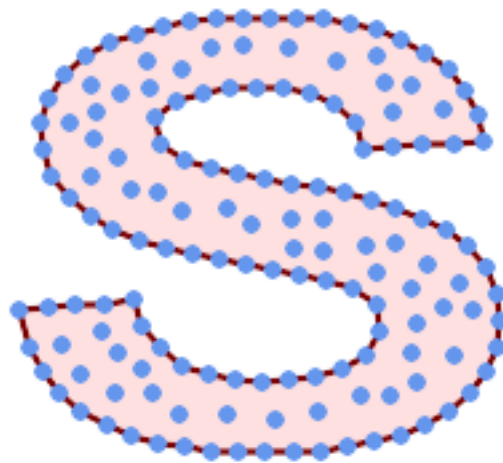
Technically, the `param_pctconvex` determines a length as a fraction of the difference between the longest and shortest edges in the Delaunay Triangulation of the input points. Edges longer than this length are "eroded" from the triangulation. The triangles remaining form the concave hull.

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\\_Collect](#) (e.g. `ST_ConcaveHull( ST_Collect( geom ), 0.80)`).

2.0.0 [\[Changelog\]](#).

Enhanced: 3.3.0, GEOS native implementation enabled for GEOS 3.11+



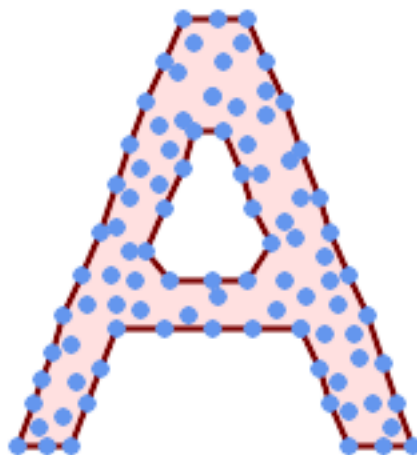
*Concave Hull of a MultiPoint*

```
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 ←

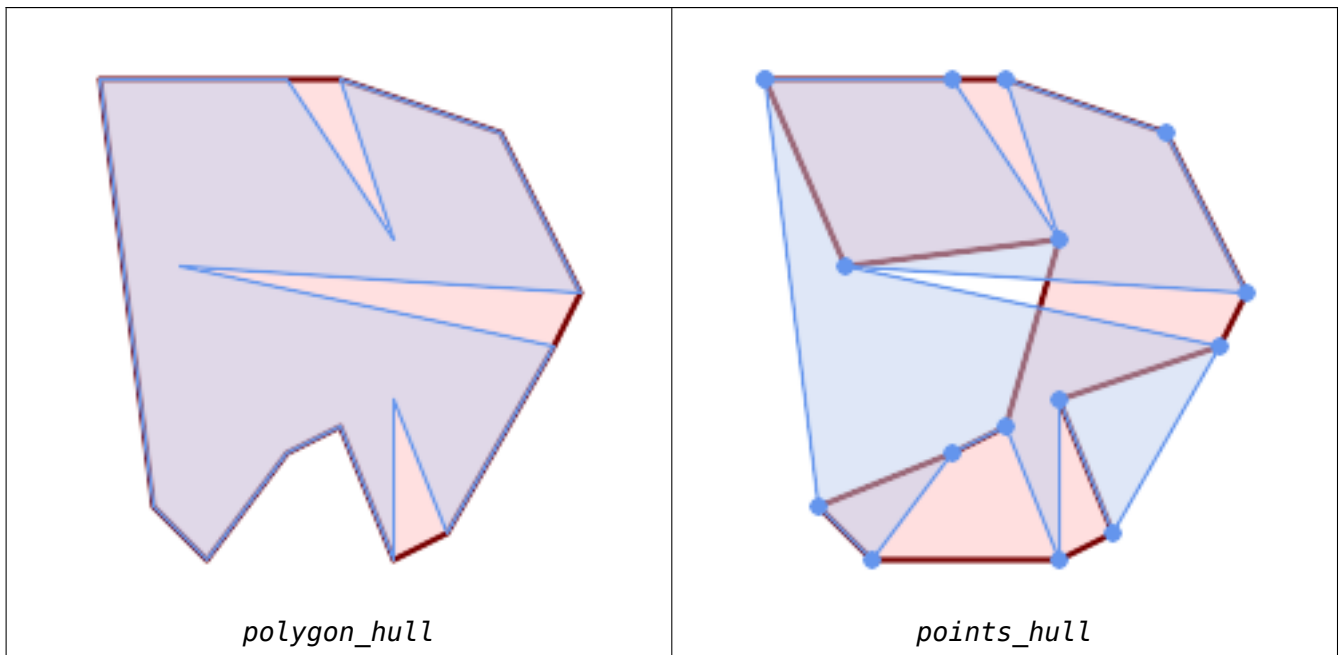
```



```

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;

```

☒☒

[ST\\_ConvexHull](#), [ST\\_Collect](#), [ST\\_AlphaShape](#), [ST\\_OptimalAlphaShape](#)

### 7.14.6 ST\_ConvexHull

`ST_ConvexHull` — Computes the convex hull of a geometry.

#### Synopsis

geometry **ST\_ConvexHull**(geometry geomA);

## ST\_ConvexHull

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.

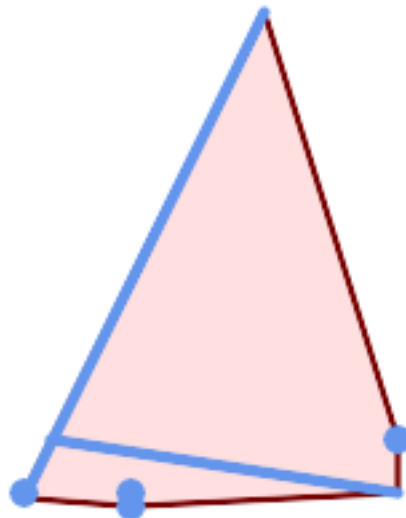
In the general case the convex hull is a Polygon. The convex hull of two or more collinear points is a two-point LineString. The convex hull of one or more identical points is a Point.

This is not an aggregate function. To compute the convex hull of a set of geometries, use **ST\_Collect** to aggregate them into a geometry collection (e.g. **ST\_ConvexHull(ST\_Collect(geom))**).

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.

## Diagram



*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;
```

国国

**ST\_Collect, 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);

国国

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

GEOS 国国国国国

2.1.0 国国国国国国国国国国国国.

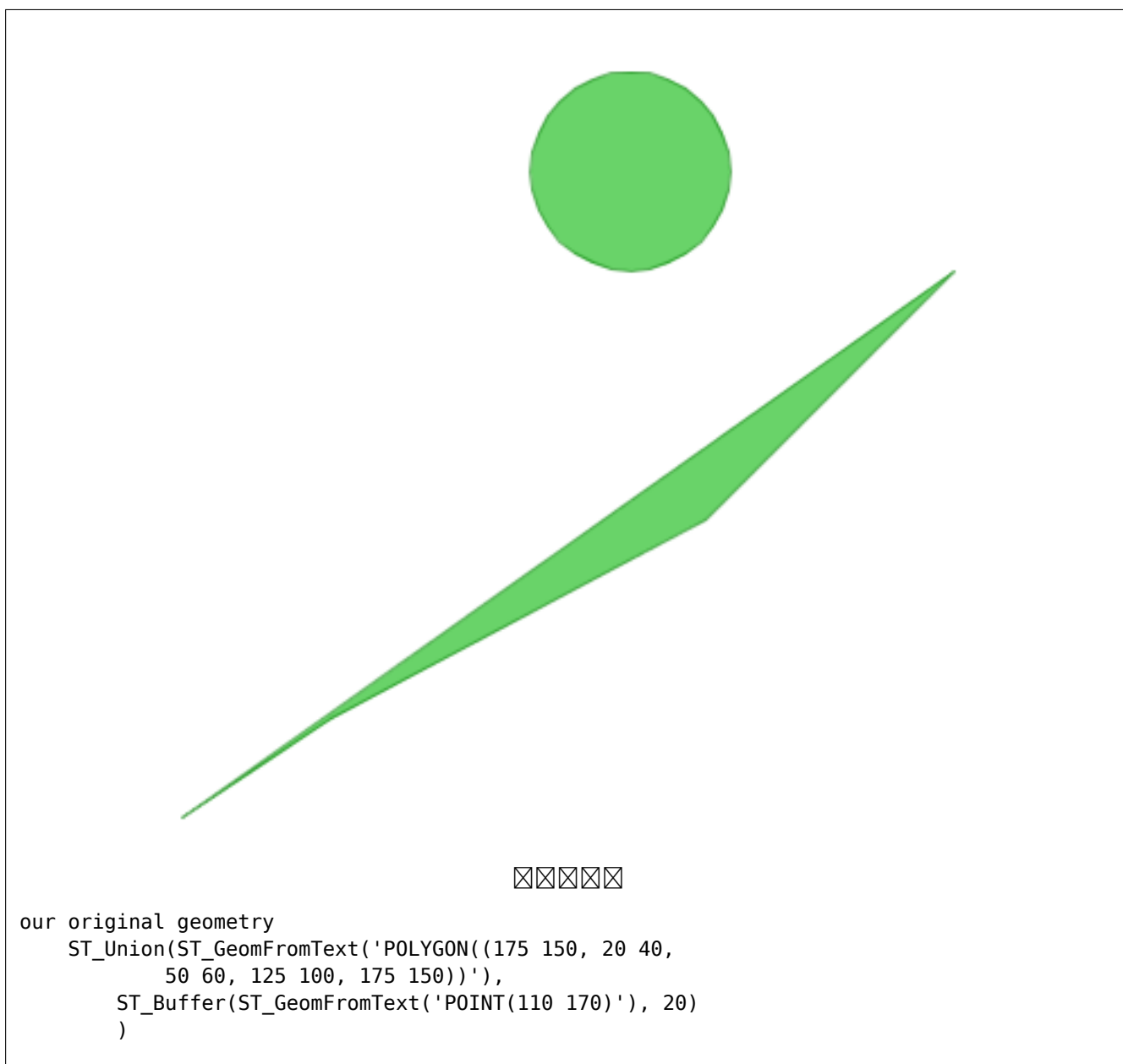


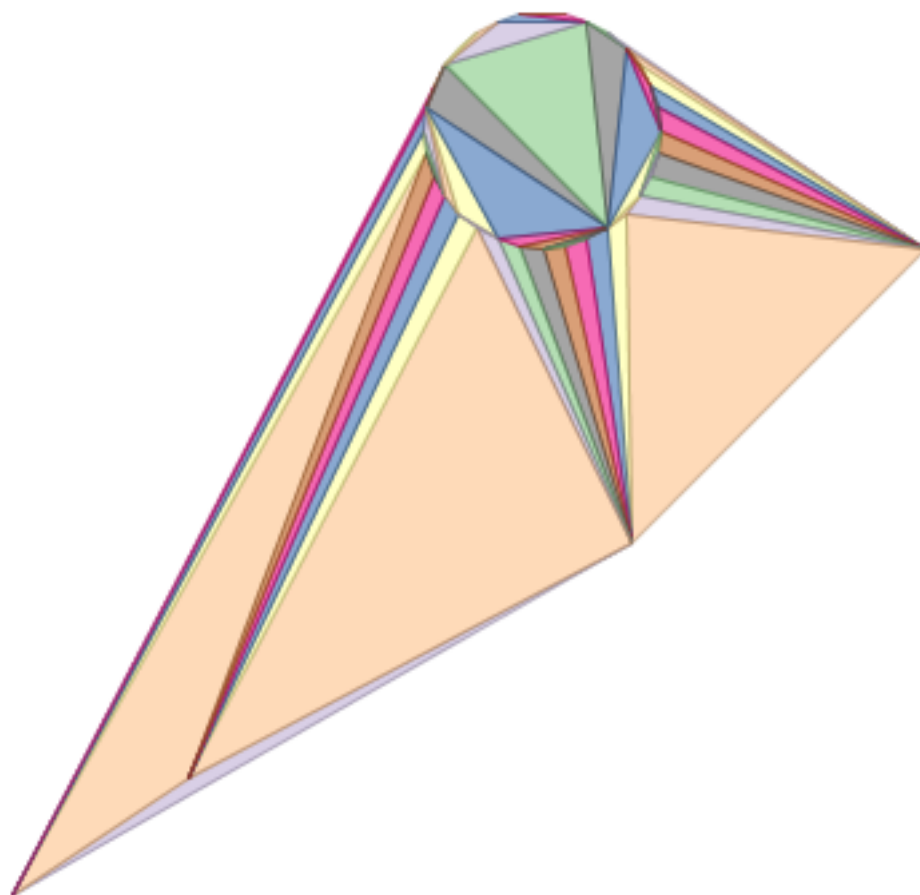
This function supports 3d and will not drop the z-index.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

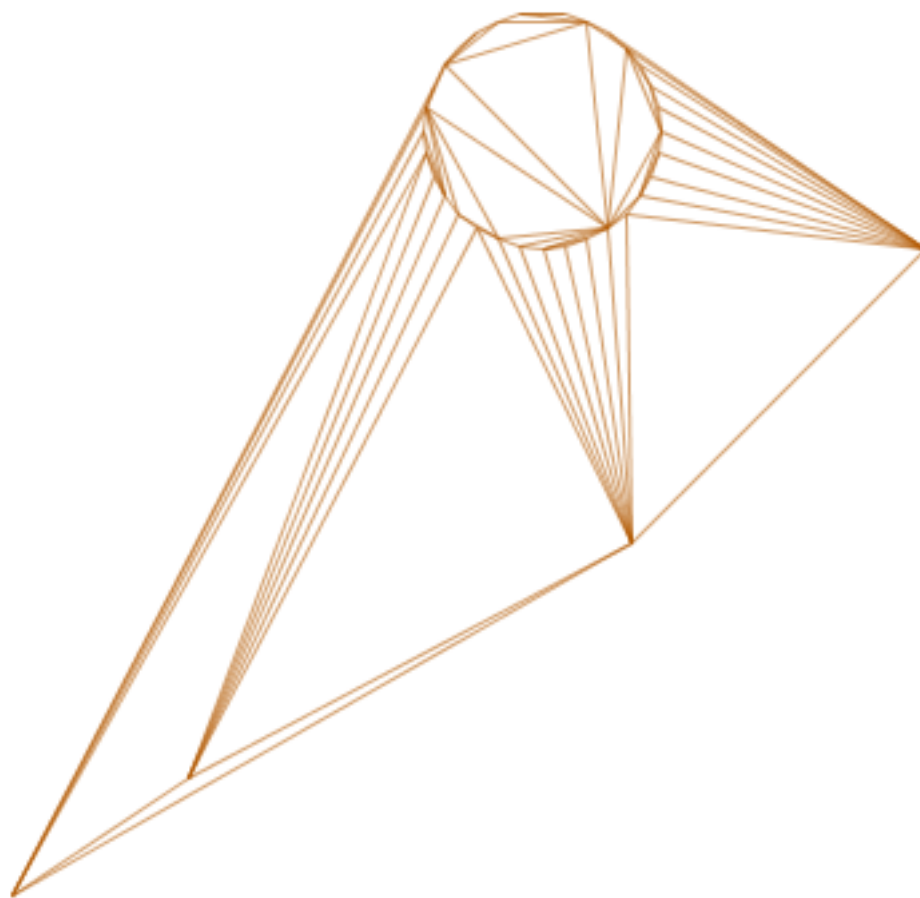
国国





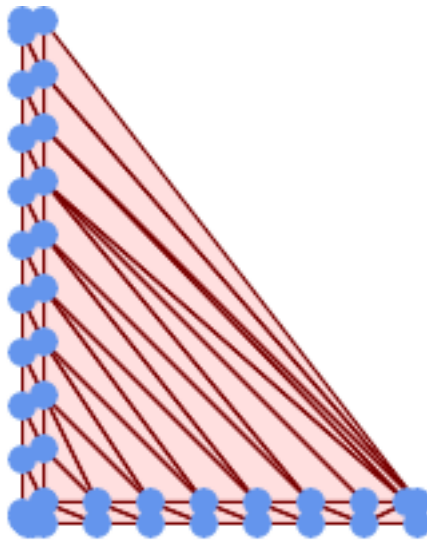
`ST_DelaunayTriangles`: `geometries overlaid multilinestring triangles`

```
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;
```



-- 繁體中文

```
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;
```



```
-- 55 45
```

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)))
```

☒☒

[ST\\_VoronoiPolygons](#), [ST\\_TriangulatePolygon](#), [ST\\_ConstrainedDelaunayTriangles](#), [ST\\_VoronoiLines](#), [ST\\_Con](#)

### 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);

☒☒

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



☐☐

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)
```

☐☐

[ST\\_SetEffectiveArea](#), [ST\\_SimplifyVW](#)

### 7.14.9 ST\_GeneratePoints

**ST\_GeneratePoints** — Generates a multipoint of random points contained in a Polygon or MultiPolygon.

#### Synopsis

geometry **ST\_GeneratePoints**(geometry g, integer npoints, integer seed = 0);

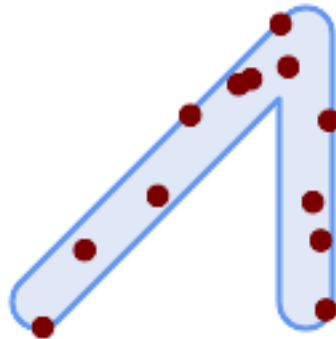
☐☐

**ST\_GeneratePoints** generates a multipoint consisting of 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.

2.3.0 文档.

Enhanced: 3.0.0, added seed parameter

☒☒



*Generated a multipoint consisting of 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;
```

Given a table of polygons *s*, return 12 individual points per polygon. Results will be different each time you run.

```
SELECT s.id, dp.path[1] AS pt_id, dp.geom
FROM s, ST_DumpPoints(ST_GeneratePoints(s.geom,12)) AS dp;
```

☒☒

## ST\_DumpPoints

### 7.14.10 ST\_GeometricMedian

ST\_GeometricMedian — ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒ (median) ☒☒☒☒☒☒.

#### Synopsis

geometry **ST\_GeometricMedian** ( geometry geom, float8 tolerance = NULL, int max\_iter = 10000, boolean fail\_if\_not\_converged = false);

ST\_GeometricMedian

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.

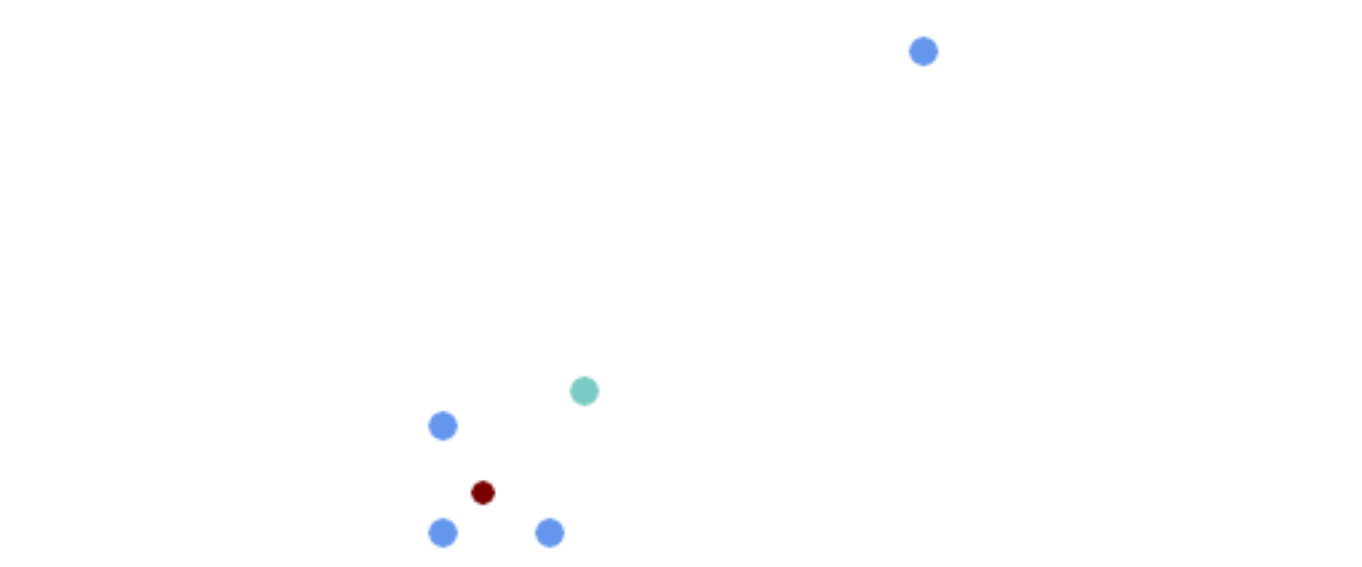
If present, the input point M values are interpreted as their relative weights.

2.3.0 Added support for M coordinates.

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.

Diagram



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)

☒☒

**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);
```

☒☒

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

GEOS ☒☒☒☒☒

Enhanced: 3.3.0 accept a directed parameter.

Requires GEOS >= 3.11.0 to use the directed parameter.

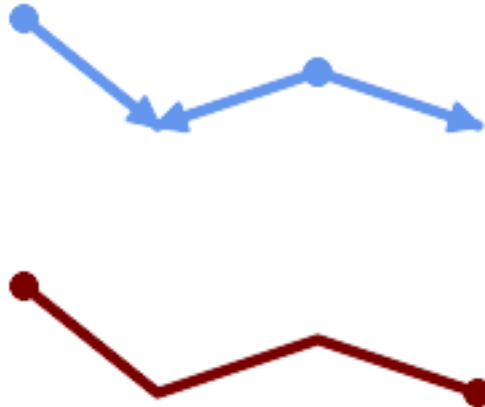
1.1.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒.



#### Warning

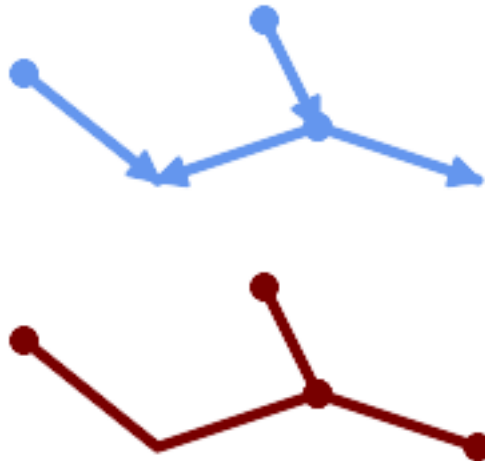
This function strips the M dimension.

☒☒



*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)
```

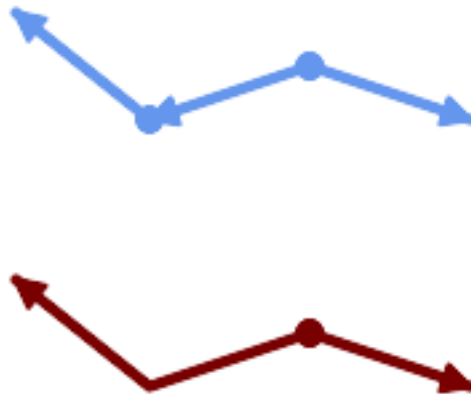


*Lines are not merged across intersections with degree > 2.*

```
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)
```



## ST Segmentize, ST LineSubstring

### 7.14.12 ST\_MaximumInscribedCircle

ST MaximumInscribedCircle — .

## Synopsis

```
(geometry, geometry, double precision) ST_MaximumInscribedCircle(geometry geom);
```

⌂⌂

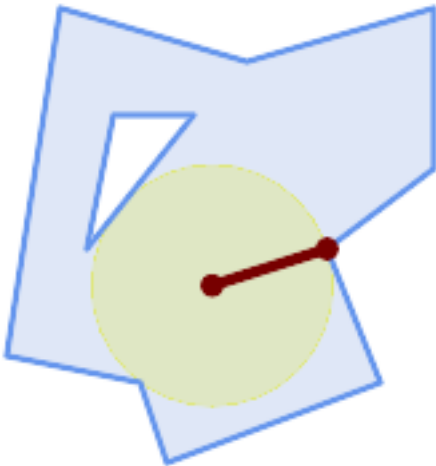
Finds the largest circle that is contained within a (multi)polygon, or which does not overlap any lines and points. Returns a record with fields:

- 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.

⌂⌂

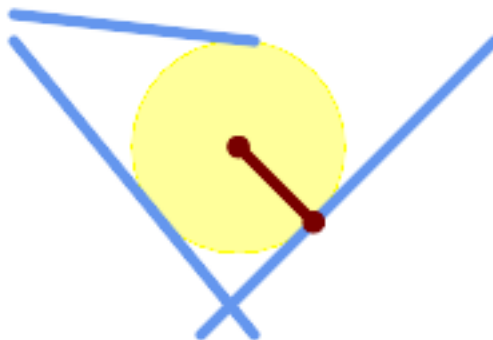


Maximum inscribed circle of a polygon. Center, nearest point, and radius are returned.

```
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.*

☒☒

[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);

☒☒

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](#).

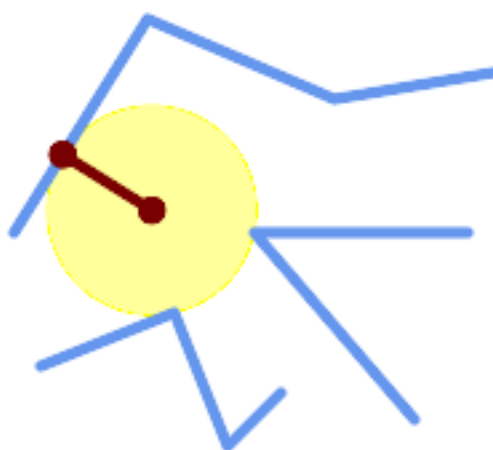
Availability: 3.4.0.

Requires GEOS >= 3.9.0.



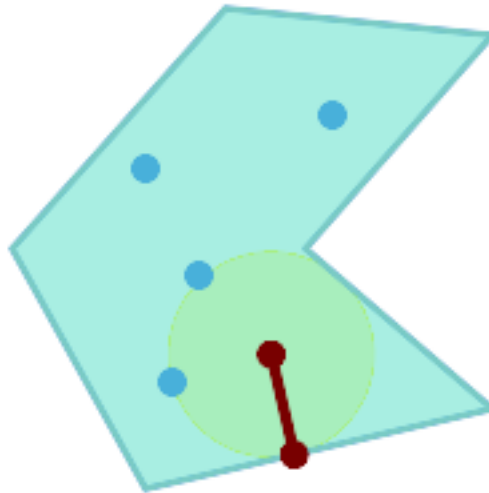
☒☒

```
SELECT radius,
       center,
       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,
       center,
       nearest
FROM ST_LargestEmptyCircle(
    ST_Collect(
        'MULTIPOINT ((70 50), (60 130), (130 150), (80 90))'::geometry,
        'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'::geometry) ←
        ,
        'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'::geometry
    );
```



*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.*



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);
```



Returns the smallest circle polygon that contains a geometry.

### Note

[illegible]

Use with **ST\_Collect** 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**.

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (Roeck) XXXXXXXXXXXXXXXX.

GEOS ☐☐☐☐☐

1.4.0 ☒☒☒☒☒☒☒☒☒☒☒.

圖 10-1

```
SELECT d.disease_type,
       ST_MinimumBoundingCircle(ST_Collect(d.geom)) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```

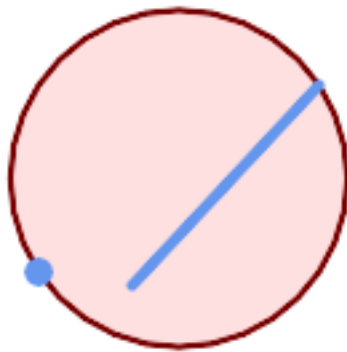


圖 10-2 顯示了 ST\_MinimumBoundingCircle 函數的輸出結果。圖中顯示了一個由多個點組成的集合，以及由這些點構成的最小外接圓。

```
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))
```

圖 10-3

ST\_Collect, 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);

**SQL**

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\_Collect** 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**.

2.3.0 **SQL**.

**SQL**

```
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 |

**SQL**

**ST\_Collect**, **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 );

**SQL**

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.

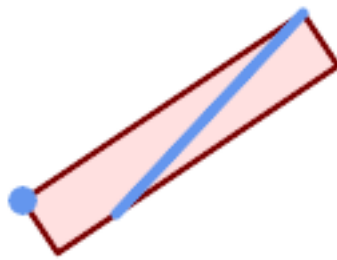
Availability: 2.5.0.

Requires GEOS >= 3.6.0.

☒☒

```
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))
```

☒☒

**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=“);



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.

[illegible]

Note that output may be a MULTILINESTRING or EMPTY for some jigsaw-shaped input geometries.

[illegible]

- 'quad\_segs=#' : □□□ (quarter circle) □□□□□□□□□□□□□□□□ (□□□□ 8)
- 'join=round|mitre|bevel' : □□□□□ (□□□□" □□ (round)"). ' □□□ (mitre)' □□□□□ □□□ (miter)' □□□□□□.
- 'mitre\_limit=#.#' : □□□□□□□ (□□□□□□□□□□□□□□□). 'mitre\_limit' □□□□□ 'miter\_limit' □□□□□□.

GEOS ☒ ☒ ☒ ☒ ☒

Behavior changed in GEOS 3.11 so offset curves now have the same direction as the input line, for both positive and negative offsets.

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.

[illegible]

```
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;
```



国国 15, 国国国国国'quad\_segs=4 join=round'  
国国国国国 15 国国

```
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)
```

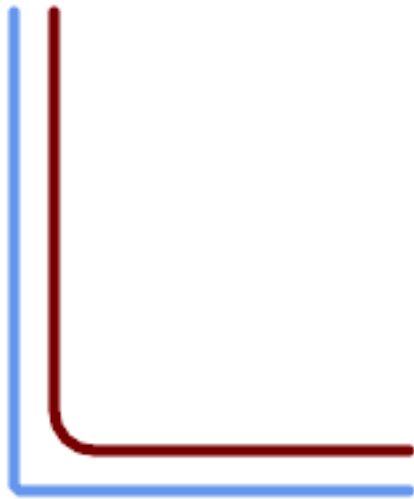


国国 -15, 国国国国国'quad\_segs=4  
join=round' 国国国国国 -15 国国

```
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

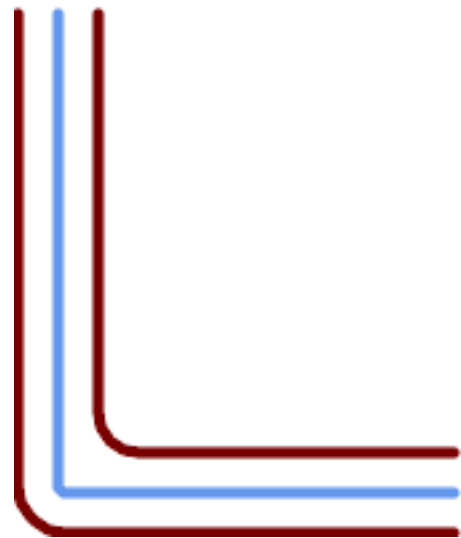
```
LINESTRING(31 195,31 31,164 31)
```



繁體中文，繁體中文  
繁體中文。  $-30 + 15 = -15$   
繁體中文。

```
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)
```



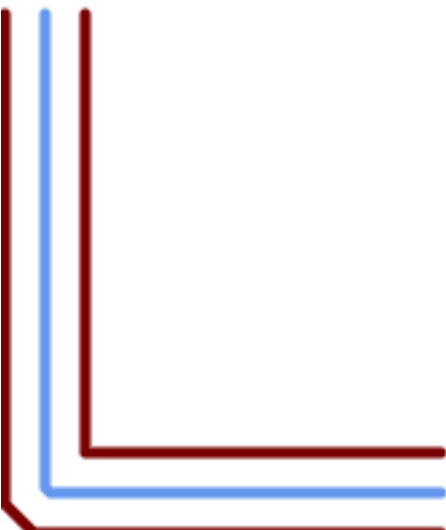
繁體中文，繁體中文 15 繁體中文，繁體  
繁體中文。 繁體中文。

```
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))
```



|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  <p>15, 'quad_segs=4<br/>join=round'</p> <pre>SELECT ST_AsText(ST_OffsetCurve(↵<br/>  ST_GeomFromText(↵<br/>'LINESTRING(164 16,144 16,124 16,104 ↵<br/>  16,84 16,64 16,↵<br/>  44 16,24 16,20 16,18 16,17 17,↵<br/>  16 18,16 20,16 40,16 60,16 80,16 100,↵<br/>  16 120,16 140,16 160,16 180,16 195)') ↵<br/>  ,↵<br/>    15, 'quad_segs=4 join=bevel'));</pre> <p>output</p> <pre>LINESTRING(164 1,18 1,7.39339828220179 ↵<br/>  5.39339828220179,↵<br/>  5.39339828220179 7.39339828220179,1 ↵<br/>  18,1 195)</pre> |  <p>15, -15. join=mitre mitre_limit=2.1</p> <pre>SELECT ST_AsText(ST_Collect(↵<br/>  ST_OffsetCurve(geom, 15, 'quad_segs=4 ↵<br/>    join=mitre mitre_limit=2.2'),↵<br/>  ST_OffsetCurve(geom, -15, 'quad_segs ↵<br/>    =4 join=mitre mitre_limit=2.2')↵<br/>  ) )↵<br/>FROM ST_GeomFromText(↵<br/>'LINESTRING(164 16,144 16,124 16,104 ↵<br/>  16,84 16,64 16,↵<br/>  44 16,24 16,20 16,18 16,17 17,↵<br/>  16 18,16 20,16 40,16 60,16 80,16 100,↵<br/>  16 120,16 140,16 160,16 180,16 195)') ↵<br/>  As geom;</pre> <p>output</p> <pre>MULTILINESTRING((164 1,11.7867965644036 ↵<br/>  1,1 11.7867965644036,1 195),↵<br/>  (31 195,31 31,164 31))</pre> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

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);



Returns a `POINT` which is guaranteed to lie in the interior of a surface (`POLYGON`, `MULTIPOLYGON`, and `CURVEPOLYGON`). In PostGIS this function also works on line and point geometries.



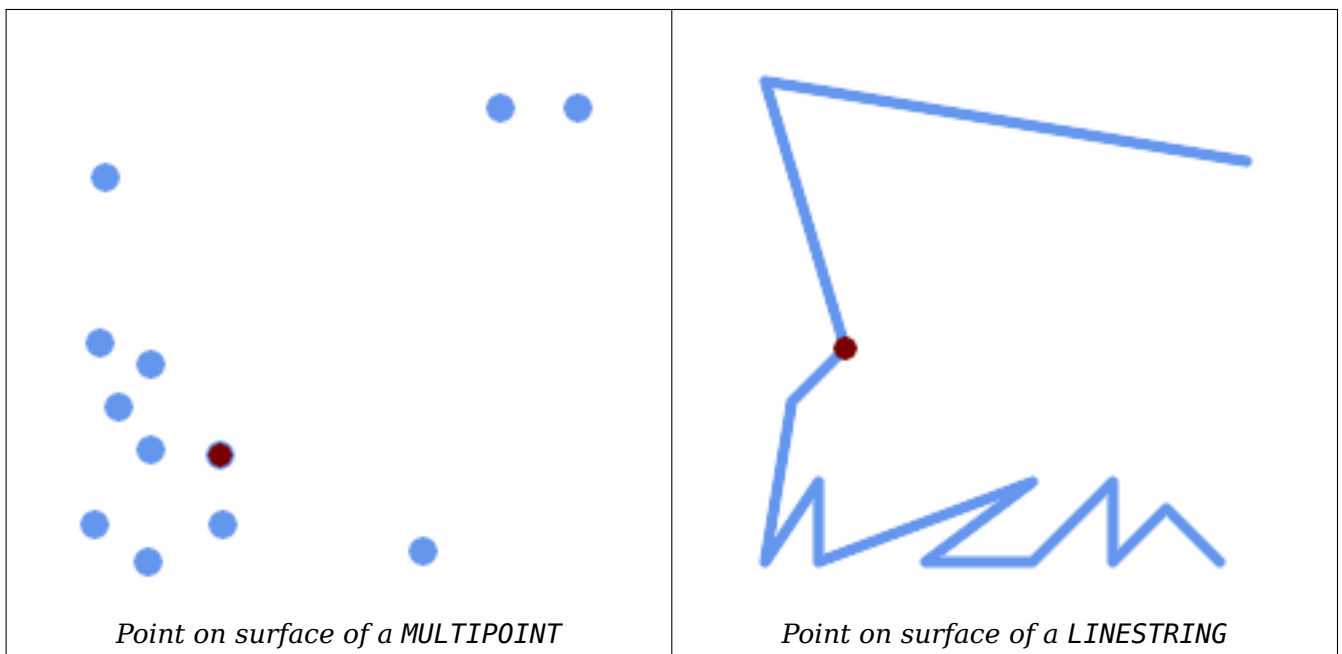
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.14.2 // s3.2.18.2

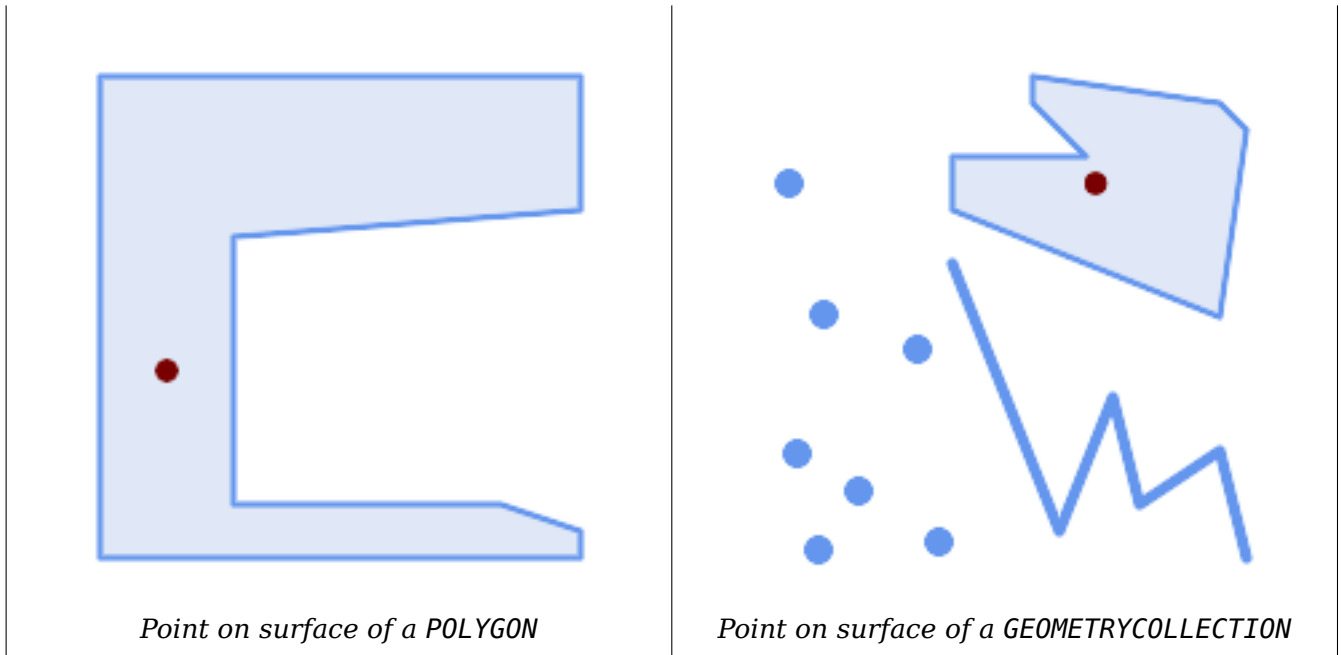


This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. The specifications define `ST_PointOnSurface` for surface geometries only. PostGIS extends the function to support all common geometry types. Other databases (Oracle, DB2, ArcSDE) seem to support this function only for surfaces. SQL Server 2008 supports all common geometry types.



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





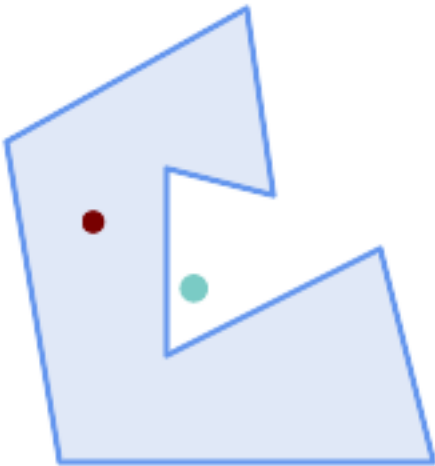
```
SELECT ST_AsText(ST_PointOnSurface('POINT(0 5) '::geometry));
-----
POINT(0 5)

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) |

🔗🔗

[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);

🔗🔗

Creates a GeometryCollection containing the polygons formed by the linework of a set of geometries. If the input linework does not form any polygons, an empty GeometryCollection is returned.

This function creates polygons covering all delimited areas. If the result is intended to form a valid polygonal geometry, use [ST\\_BuildArea](#) to prevent holes being filled.



**Note**  
The input linework must be correctly noded for this function to work properly. To ensure input is noded use [ST\\_Node](#) on the input geometry before polygonizing.

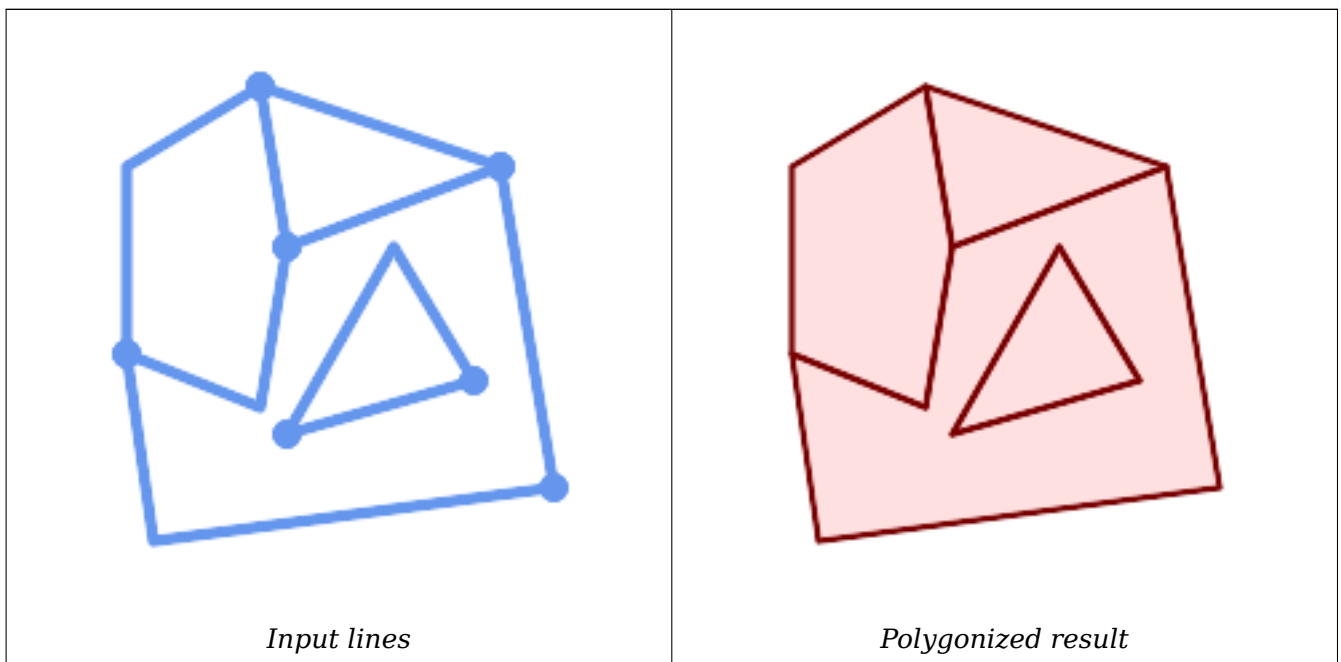
**Note**

GeometryCollections can be difficult to handle with external tools. Use **ST\_Dump** to convert the polygonized result into separate polygons.

GEOS 国国国国国

1.0.0RC1 国国国国国国国国国国国国.

国国



```
WITH data(geom) AS (VALUES
  ('LINESTRING (180 40, 30 20, 20 90)')::geometry)
, ('LINESTRING (180 40, 160 160)')::geometry)
, ('LINESTRING (80 60, 120 130, 150 80)')::geometry)
, ('LINESTRING (80 60, 150 80)')::geometry)
, ('LINESTRING (20 90, 70 70, 80 130)')::geometry)
, ('LINESTRING (80 130, 160 160)')::geometry)
, ('LINESTRING (20 90, 20 160, 70 190)')::geometry)
, ('LINESTRING (70 190, 80 130)')::geometry)
, ('LINESTRING (70 190, 160 160)')::geometry)
)
SELECT ST_AsText( ST_Polygonize( geom ))
FROM data;
```

```
-----
GEOMETRYCOLLECTION (POLYGON ((180 40, 30 20, 20 90, 70 70, 80 130, 160 160, 180 40), (150 80, 120 130, 80 60, 150 80)),
  POLYGON ((20 90, 20 160, 70 190, 80 130, 70 70, 20 90)),
  POLYGON ((160 160, 80 130, 70 190, 160 160)),
  POLYGON ((80 60, 120 130, 150 80, 80 60)))
```

Polygonizing a table of linestrings:

```

SELECT ST_AsEWKT(ST_Polygonize(geom_4269)) As geomtextrep
FROM (SELECT geom_4269 FROM ma.suffolk_edges) As foo;

-----
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)))

--Use ST_Dump to dump out the polygonize geoms into individual polygons
SELECT ST_AsEWKT((ST_Dump(t.polycoll)).geom) AS geomtextrep
FROM (SELECT ST_Polygonize(geom_4269) AS polycoll
      FROM (SELECT geom_4269 FROM ma.suffolk_edges)
      As foo) AS t;

-----
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))

```

☒☒

[ST\\_BuildArea](#), [ST\\_Dump](#), [ST\\_Node](#)

### 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);

☒☒

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.

图

```
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)))
```

图

**ST\_SnapToGrid**, **ST\_Simplify**, **ST\_SimplifyVW**

### 7.14.21 ST\_SharedPaths

**ST\_SharedPaths** — 返回两个几何体共享的路径。

#### Synopsis

geometry **ST\_SharedPaths**(geometry lineal1, geometry lineal2);

图

该函数返回两个几何体共享的路径。如果两个几何体没有共享的路径，则返回空几何体。如果两个几何体是点，则返回空几何体。如果两个几何体是线，则返回共享的线段。如果两个几何体是面，则返回共享的面。

GEOS 版本

2.0.0 版本。

图: 共享路径

|                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                                                                                                                                                                                                                                                                  |
| 简体中文                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|                                                                                                                                                                                                                                                                                                                                                                 |
| 简体中文                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <pre>SELECT ST_AsText(<br/>  ST_SharedPaths(<br/>    ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),<br/>      (51 150,101 150,76 175,51 150))'),<br/>    ST_GeomFromText('LINESTRING(151 100,126 156.25,126 125,90 161, 76 175)')<br/>  )<br/>  ) As wkt<br/><br/>      wkt<br/>-----<br/>GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125),<br/>  (101 150,90 161),(90 161,76 175)),MULTILINESTRING EMPTY)</pre> |



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)))
```

☒☒

[ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

## 7.14.22 ST\_Simplify

**ST\_Simplify** — Returns a simplified representation of a geometry, using the Douglas-Peucker algorithm.

### Synopsis

```
geometry ST_Simplify(geometry geom, float tolerance);
geometry ST_Simplify(geometry geom, float tolerance, boolean preserveCollapsed);
```

☒☒

Computes a simplified representation of a geometry using the [Douglas-Peucker algorithm](#). The simplification tolerance is a distance value, in the units of the input SRS. Simplification removes vertices which are within the tolerance distance of the simplified linework. The result may not be valid even if the input is.

The function can be called with any kind of geometry (including GeometryCollections), but only line and polygon elements are simplified. Endpoints of linear geometry are preserved.

The `preserveCollapsed` flag retains small geometries that would otherwise be removed at the given tolerance. For example, if a 1m long line is simplified with a 10m tolerance, when `preserveCollapsed` is true the line will not disappear. This flag is useful for rendering purposes, to prevent very small features disappearing from a map.



### Note

The returned geometry may lose its simplicity (see [ST\\_IsSimple](#)), topology may not be preserved, and polygonal results may be invalid (see [ST\\_IsValid](#)). Use [ST\\_SimplifyPreserveTopology](#) to preserve topology and ensure validity.



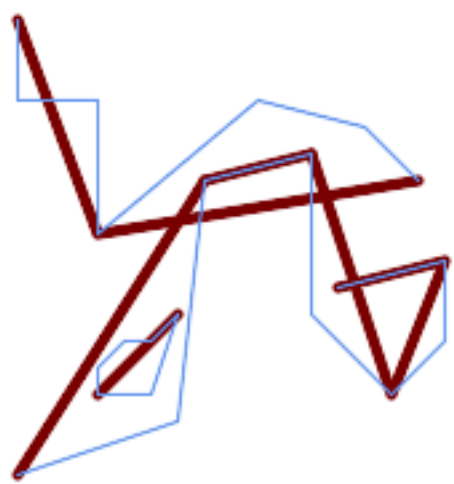
**Note** This function does not preserve boundaries shared between polygons. Use **ST\_CoverageSimplify** if this is required.

1.2.2

```
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 t;
```

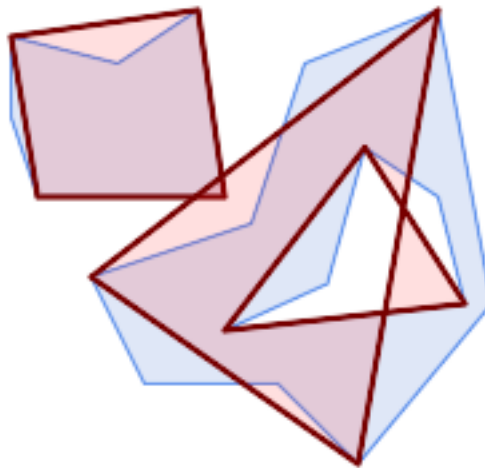
| np_before | np01_notbadcircle | np05_notquitecircle | np1_octagon | np10_triangle | np100_geometrygoesaway |
|-----------|-------------------|---------------------|-------------|---------------|------------------------|
| 49        | 33                | 17                  | 9           | 4             | t                      |

Simplifying a set of lines. Lines may intersect after simplification.



```
SELECT ST_Simplify(
  'MULTILINESTRING ((20 180, 20 150, 50 150, 50 100, 110 150, 150 140, 170 120), (20 10, 80 10, 80 30, 90 120), (90 120, 130 130), (130 130, 130 70, 160 40, 180 60, 180 90, 140 80), (50 40, 70 40, 80 70, 70 60, 60 60, 50 50, 50 40))',
  40);
```

Simplifying a MultiPolygon. Polygonal results may be invalid.



```
SELECT ST_Simplify(
  'MULTIPOLYGON (((90 110, 80 180, 50 160, 10 170, 10 140, 20 110, 90 110)), ((40 80, 100 80, 100 120 160, 170 180, 190 70, 140 10, 110 40, 60 40, 40 80), (180 70, 170 110, 142.5 128.5, 128.5 77.5, 90 60, 180 70)))',
  40);
```

☒☒

[ST\\_IsSimple](#), [ST\\_SimplifyPreserveTopology](#), [ST\\_SimplifyVW](#), [ST\\_CoverageSimplify](#), [Topology](#) [ST\\_Simplify](#)

### 7.14.23 ST\_SimplifyPreserveTopology

**ST\_SimplifyPreserveTopology** — Returns a simplified and valid representation of a geometry, using the Douglas-Peucker algorithm.

#### Synopsis

geometry **ST\_SimplifyPreserveTopology**(geometry geom, float tolerance);

☒☒

Computes a simplified representation of a geometry using a variant of the [Douglas-Peucker algorithm](#) which limits simplification to ensure the result has the same topology as the input. The simplification tolerance is a distance value, in the units of the input SRS. Simplification removes vertices which are within the tolerance distance of the simplified linework, as long as topology is preserved. The result will be valid and simple if the input is.

The function can be called with any kind of geometry (including GeometryCollections), but only line and polygon elements are simplified. For polygonal inputs, the result will have the same number of rings (shells and holes), and the rings will not cross. Ring endpoints may be simplified. For linear inputs, the result will have the same number of lines, and lines will not intersect if they did not do so in the original geometry. Endpoints of linear geometry are preserved.



**Note** This function does not preserve boundaries shared between polygons. Use `ST_CoverageSimplify` if this is required.

GEOS

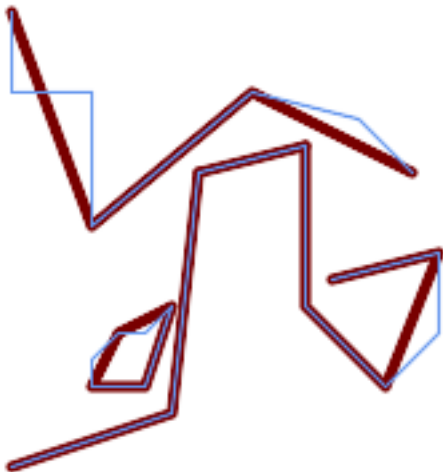
1.3.3

For the same example as `ST_Simplify`, `ST_SimplifyPreserveTopology` prevents oversimplification. The circle can at most become a square.

```
SELECT  ST_Npoints geom AS np_before,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.1)) AS np01_notbadcircle,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.5)) AS np05_notquitecircle,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 1))   AS np1_octagon,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 10))  AS np10_square,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 100)) AS np100_stillsquare
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) AS geom) AS t;
```

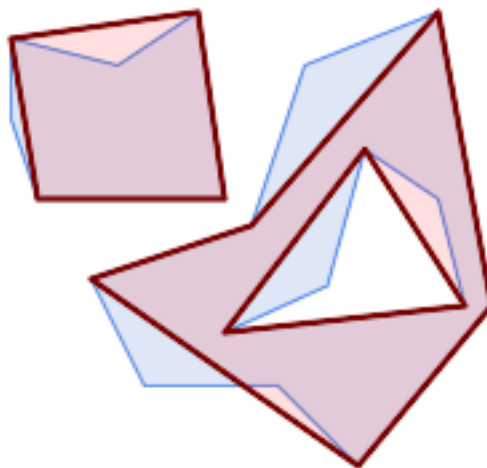
| np_before | np01_notbadcircle | np05_notquitecircle | np1_octagon | np10_square |  |
|-----------|-------------------|---------------------|-------------|-------------|--|
| 49        | 33                | 17                  | 9           | 5           |  |

Simplifying a set of lines, preserving topology of non-intersecting lines.



```
SELECT ST_SimplifyPreserveTopology(
  'MULTILINESTRING ((20 180, 20 150, 50 150, 50 100, 110 150, 150 140, 170 120), (20 10, 80 10, 80 30, 90 120), (90 120, 130 130), (130 130, 130 70, 160 40, 180 60, 180 90, 140 80), (50 40, 70 40, 80 70, 70 60, 60 60, 50 50, 50 40))',
  40);
```

Simplifying a MultiPolygon, preserving topology of shells and holes.



```
SELECT ST_SimplifyPreserveTopology(
  'MULTIPOLYGON (((90 110, 80 180, 50 160, 10 170, 10 140, 20 110, 90 110)), ((40 80, 100 80, 100 120, 160 170, 180 190, 70 140, 10 110, 40 60, 40 40, 80 40), (180 70, 170 110, 142.5 128.5, 128.5 77.5, 90 60, 180 70)))',
  40);
```

☒☒

[ST\\_Simplify](#), [ST\\_SimplifyVW](#), [ST\\_CoverageSimplify](#)

#### 7.14.24 ST\_SimplifyPolygonHull

**ST\_SimplifyPolygonHull** — Computes a simplified topology-preserving outer or inner hull of a polygonal geometry.

##### Synopsis

geometry **ST\_SimplifyPolygonHull**(geometry param\_geom, float vertex\_fraction, boolean is\_outer = true);

☒☒

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.

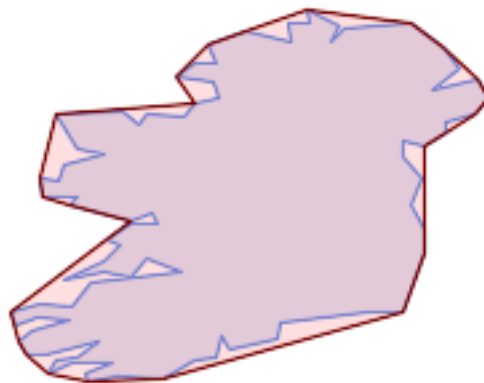
The simplification process operates by progressively removing concave corners that contain the least amount of area, until the vertex count target is reached. It prevents edges from crossing, so the result is always a valid polygonal geometry.

To get better results with geometries that contain relatively long line segments, it might be necessary to "segmentize" the input, as shown below.

GEOS

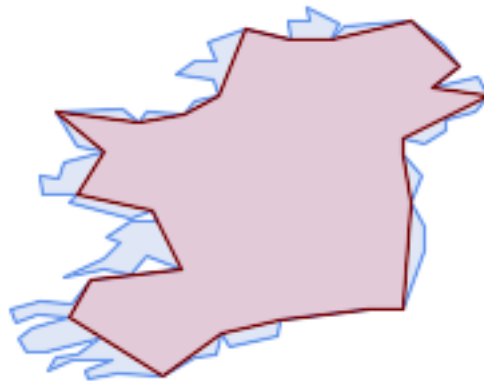
Availability: 3.3.0.

Requires GEOS >= 3.11.0.



*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);
```

☒☒

[ST\\_ConvexHull](#), [ST\\_SimplifyVW](#), [ST\\_ConcaveHull](#), [ST\\_Segmentize](#)

### 7.14.25 ST\_SimplifyVW

`ST_SimplifyVW` — Returns a simplified representation of a geometry, using the Visvalingam-Whyatt algorithm

#### Synopsis

geometry **ST\_SimplifyVW**(geometry geom, float tolerance);

Returns a simplified representation of a geometry using the **Visvalingam-Whyatt algorithm**. The simplification tolerance is an area value, in the units of the input SRS. Simplification removes vertices which form "corners" with area less than the tolerance. The result may not be valid even if the input is.

The function can be called with any kind of geometry (including GeometryCollections), but only line and polygon elements are simplified. Endpoints of linear geometry are preserved.



**Note**

The returned geometry may lose its simplicity (see `ST_IsSimple`), topology may not be preserved, and polygonal results may be invalid (see `ST_IsValid`). Use `ST_SimplifyPreserveTopology` to preserve topology and ensure validity. `ST_CoverageSimplify` also preserves topology and validity.



**Note**

This function does not preserve boundaries shared between polygons. Use `ST_CoverageSimplify` if this is required.



**Note**

3 , , .

2.2.0 .

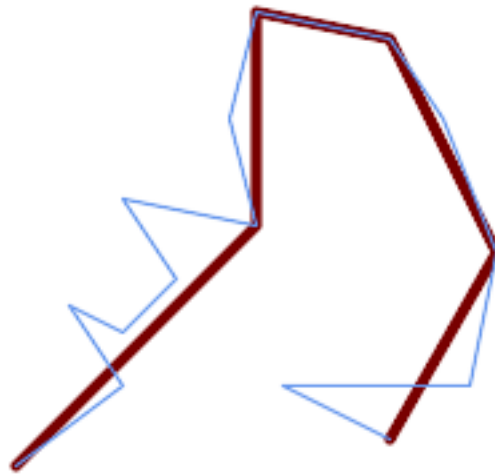
A LineString is simplified with a minimum-area tolerance of 30.

```
SELECT ST_AsText(ST_SimplifyVW(geom,30)) simplified
FROM (SELECT 'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)'::geometry AS geom) AS t;

simplified
-----
LINESTRING(5 2,7 25,10 10)
```

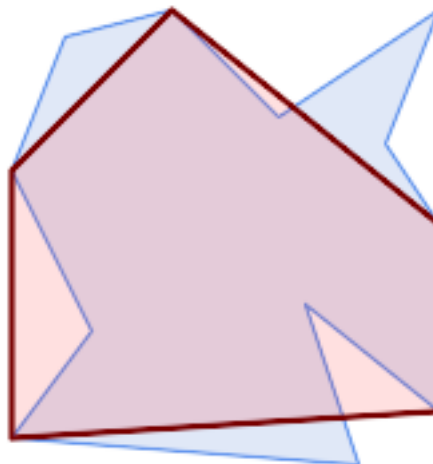
Simplifying a line.





```
SELECT ST_SimplifyVW(
  'LINESTRING (10 10, 50 40, 30 70, 50 60, 70 80, 50 110, 100 100, 90 140, 100 180, 150
    170, 170 140, 190 90, 180 40, 110 40, 150 20)',
  1600);
```

Simplifying a polygon.



```
SELECT ST_SimplifyVW(
  'MULTIPOLYGON (((90 110, 80 180, 50 160, 10 170, 10 140, 20 110, 90 110)), ((40 80, 100
    100, 120 160, 170 180, 190 70, 140 10, 110 40, 60 40, 40 80), (180 70, 170 110, 142.5
    128.5, 128.5 77.5, 90 60, 180 70))))',
  40);
```

☒☒

[ST\\_SetEffectiveArea](#), [ST\\_Simplify](#), [ST\\_SimplifyPreserveTopology](#), [ST\\_CoverageSimplify](#), [Topology](#) [ST\\_Simpl](#)

### 7.14.26 ST\_SetEffectiveArea

`ST_SetEffectiveArea` — Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.

## Synopsis

geometry **ST\_SetEffectiveArea**(geometry geom, float threshold = 0, integer set\_area = 1);

几何

返回几何-有效面积。如果 threshold 为 0，则返回 M 值。如果 threshold 不为 0，则返回“有效”面积，即几何中所有面积大于 threshold 的区域的并集。

如果 threshold 为 0，则返回 M 值。如果 threshold 不为 0，则返回“有效”面积，即几何中所有面积大于 threshold 的区域的并集。

如果 threshold 不为 0，则返回“有效”面积，即几何中所有面积大于 threshold 的区域的并集。

Note!

### Note

如果几何不是简单几何 (ST\_IsSimple 为真)。

Note!

### Note

如果 (topology) 为真，则返回 ST\_SimplifyPreserveTopology 的结果。

Note!

### Note

如果 M 值为 0，则返回 0。

Note!

### Note

如果 threshold 为 0，则返回 M 值。

2.2.0 版本中，返回的几何是有效的。

几何

返回几何-有效面积。如果 threshold 为 0，则返回 M 值。如果 threshold 不为 0，则返回“有效”面积，即几何中所有面积大于 threshold 的区域的并集。

```
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)
```

☒☒

**ST\_SimplifyVW**

### 7.14.27 ST\_TriangulatePolygon

ST\_TriangulatePolygon — Computes the constrained Delaunay triangulation of polygons

#### Synopsis

geometry **ST\_TriangulatePolygon**(geometry geom);

☒☒

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.

☒☒

Triangulation of a square.

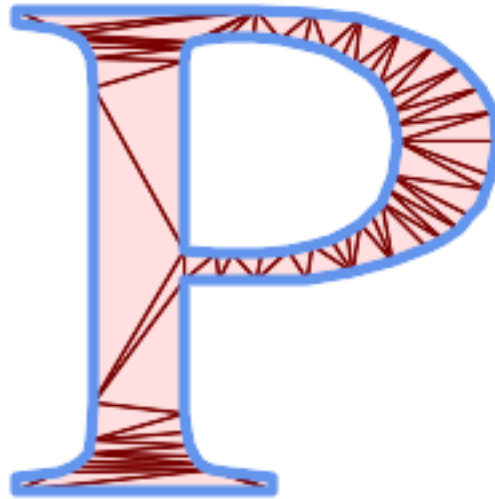
```
SELECT ST_AsText(
  ST_TriangulatePolygon('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))');

          st_astext
-----
GEOMETRYCOLLECTION(POLYGON((0 0,0 1,1 1,0 0)),POLYGON((1 1,1 0,0 0,1 1)))
```

☒☒

Triangulation of the letter P.

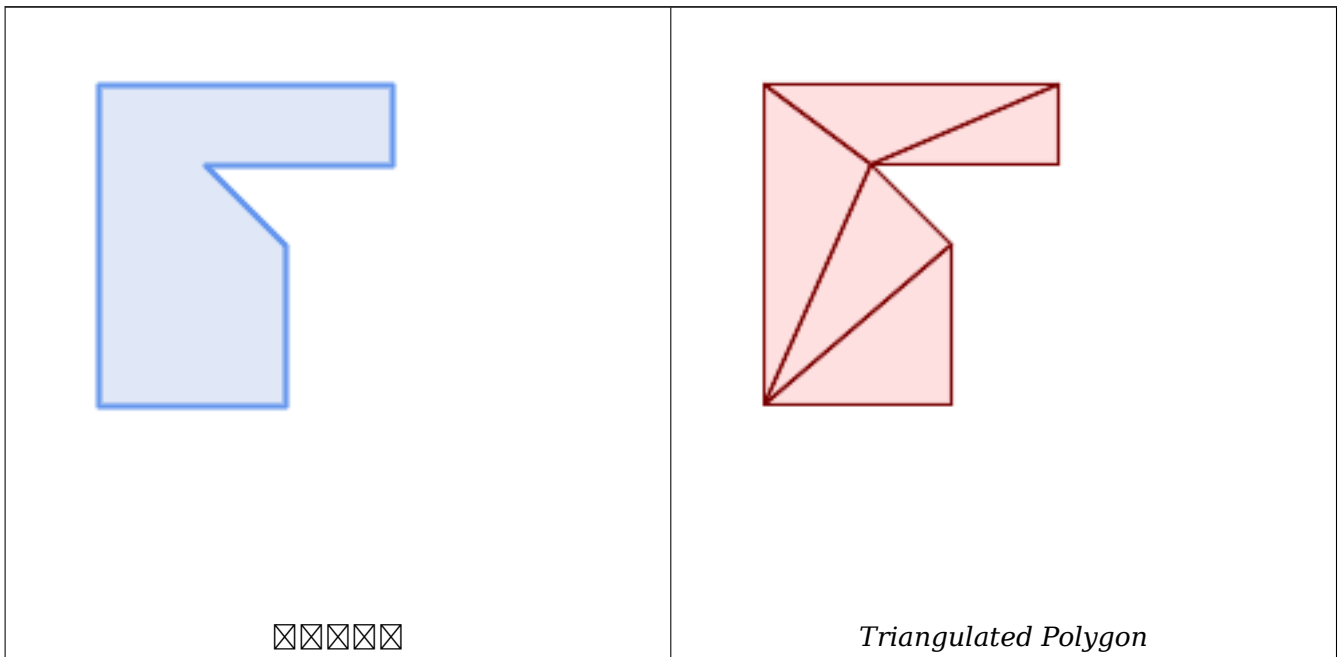
```
SELECT ST_AsText(ST_TriangulatePolygon(
  'POLYGON ((26 17, 31 19, 34 21, 37 24, 38 29, 39 43, 39 161, 38 172, 36 176, 34 179, 30 ↵
    181, 25 183, 10 185, 10 190, 100 190, 121 189, 139 187, 154 182, 167 177, 177 169, ↵
    184 161, 189 152, 190 141, 188 128, 186 123, 184 117, 180 113, 176 108, 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***Same example as ST\_Tessellate**

```
SELECT ST_TriangulatePolygon(
    'POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160, 120 160, 120 190, 10 190 ←
    ))'::geometry
);
```

ST\_AsText 繁體中文:

```
GEOMETRYCOLLECTION(POLYGON((50 160,120 190,120 160,50 160))
    ,POLYGON((10 70,80 130,80 70,10 70))
    ,POLYGON((50 160,10 70,10 190,50 160))
    ,POLYGON((120 190,50 160,10 190,120 190))
    ,POLYGON((80 130,10 70,50 160,80 130)))
```



☒☒

[ST\\_ConstrainedDelaunayTriangles](#), [ST\\_DelaunayTriangles](#), [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 );

☒☒

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.

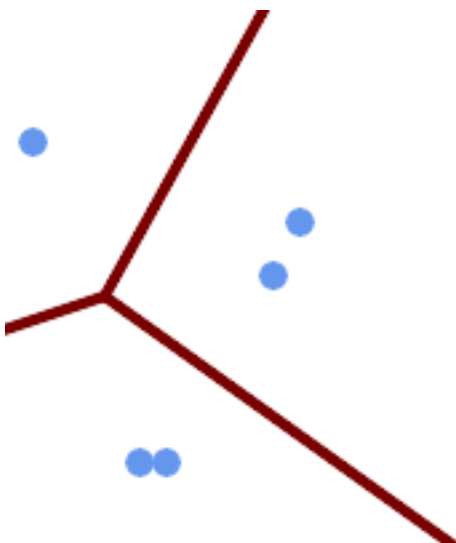
☒☒☒☒☒☒☒☒☒☒:

- **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%).

GEOS ☒☒☒☒☒

2.3.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒



*Voronoi diagram lines, with tolerance of 30 units*

```
SELECT ST_VoronoiLines(
  'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)::geometry,
  30) AS geom;
```

```
ST_AsText output
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273),(36.8181818181818 92.2727272727273,-110 43.3333333333333),(230 -45.7142857142858,36.8181818181818 92.2727272727273))
```

☒☒

[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 );

☒☒

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.

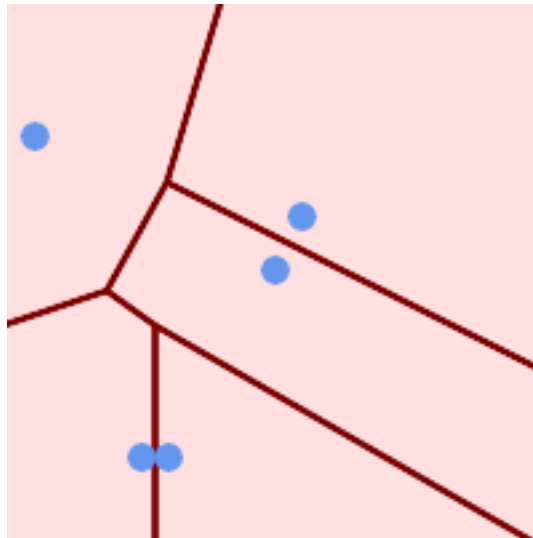
☒☒☒☒☒☒☒☒☒☒:

- **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%).

GEOS ☒☒☒☒☒

2.3.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

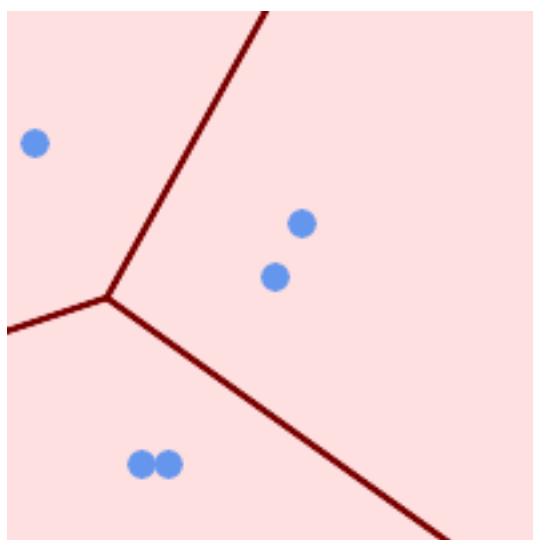


*Points overlaid on top of Voronoi diagram*

```
SELECT ST_VoronoiPolygons(
    'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120) '::geometry
) AS geom;
```

ST\_AsText output

```
GEOMETRYCOLLECTION(POLYGON((-110 43.33333333333333,-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)))
```

☒☒

[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);

☒☒

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.

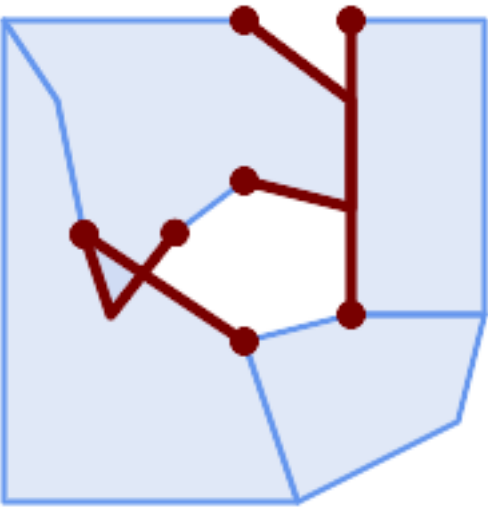
The conditions allow a valid coverage to contain holes (gaps between polygons), as long as the surrounding polygons are edge-matched. However, very narrow gaps are often undesirable. If the *tolerance* parameter is specified with a non-zero distance, edges forming narrower gaps will also be returned 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





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)                                                      |
| 4  | null                                                                              |

```
-- Test entire table for coverage validity
SELECT true = ALL (
  SELECT ST_CoverageInvalidEdges(geom) OVER () IS NULL
  FROM coverage
);
```



[ST\\_IsValid](#), [ST\\_CoverageUnion](#), [ST\\_CoverageClean](#), [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);

☒☒

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.

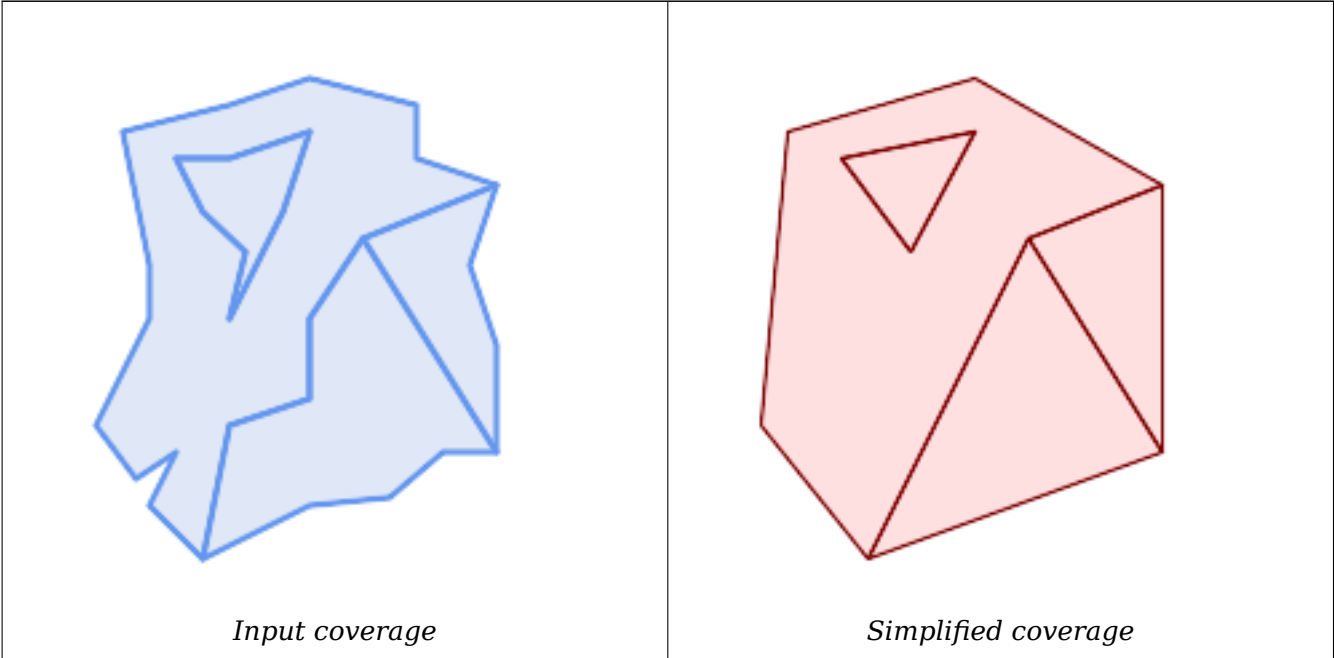
To simplify only the "internal" edges of the coverage (those that are shared by two polygons) set the *simplifyBoundary* parameter to false.



**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

☒☒



```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((160 150, 110 130, 90 100, 90 70, 60 60, 50 10, 30 30, 40 50, 25 40, 10 60, ↵
    30 100, 30 120, 20 170, 60 180, 90 190, 130 180, 130 160, 160 150), (40 160, 50 140, ↵
    66 125, 60 100, 80 140, 90 170, 60 160, 40 160))':::geometry),
```

```
(2, 'POLYGON ((40 160, 60 160, 90 170, 80 140, 60 100, 66 125, 50 140, 40 160))':: geometry),
(3, 'POLYGON ((110 130, 160 50, 140 50, 120 33, 90 30, 50 10, 60 60, 90 70, 90 100, 110 130))'::geometry),
(4, 'POLYGON ((160 150, 150 120, 160 90, 160 50, 110 130, 160 150))'::geometry)
)
SELECT id, ST_AsText(ST_CoverageSimplify(geom, 30) OVER ())
FROM coverage;
```

| id | st_astext                                                                                             |
|----|-------------------------------------------------------------------------------------------------------|
| 1  | POLYGON ((160 150, 110 130, 50 10, 10 60, 20 170, 90 190, 160 150), (40 160, 66 125, 90 170, 40 160)) |
| 2  | POLYGON ((40 160, 66 125, 90 170, 40 160))                                                            |
| 3  | POLYGON ((110 130, 160 50, 50 10, 110 130))                                                           |
| 4  | POLYGON ((160 150, 160 50, 110 130, 160 150))                                                         |



[ST\\_CoverageInvalidEdges](#), [ST\\_CoverageUnion](#), [ST\\_CoverageClean](#)

### 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);



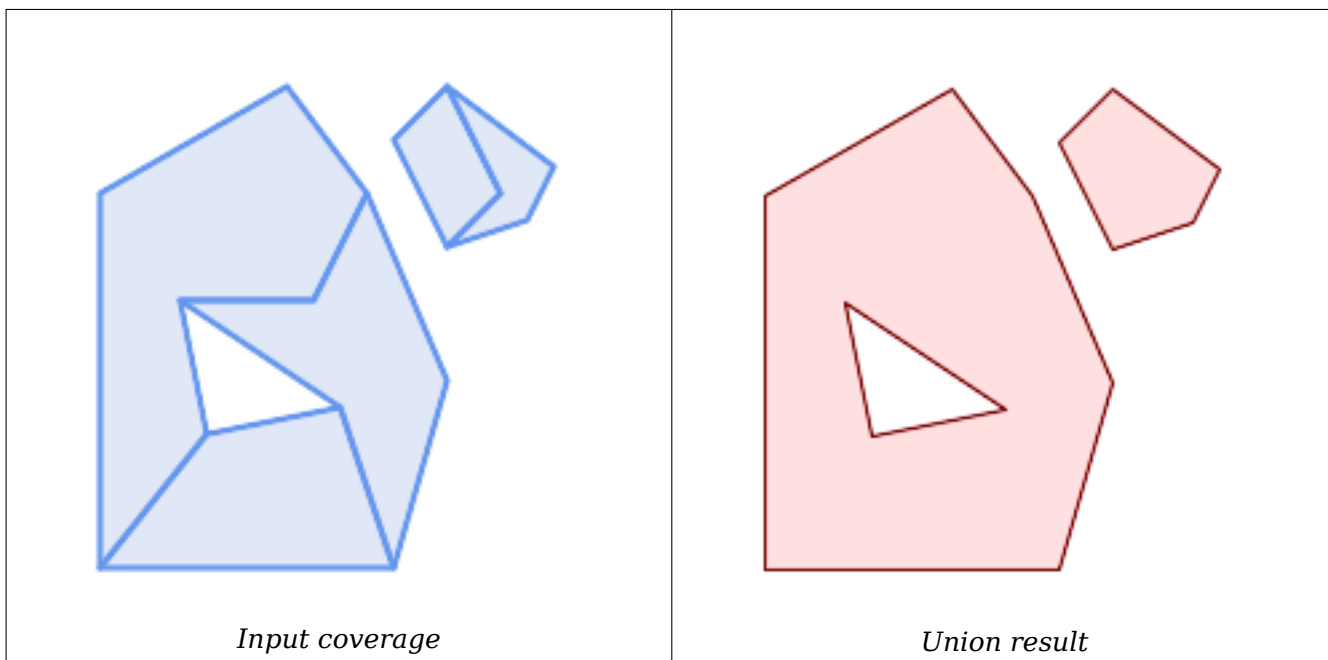
An aggregate function which unions a set of polygons forming a polygonal coverage. The result is a polygonal geometry covering the same area as the coverage. This function produces the same result as [ST\\_Union](#), but uses the coverage structure to compute the union much faster.



**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





```
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)))
```

☒☒

[ST\\_CoverageInvalidEdges](#), [ST\\_CoverageSimplify](#), [ST\\_CoverageClean](#), [ST\\_Union](#)

### 7.15.4 ST\_CoverageClean

**ST\_CoverageClean** — Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.

#### Synopsis

geometry **ST\_CoverageClean**(geometry winset geom, float8 snappingDistance = -1, float8 gapMaximumWidth = 0, text overlapMergeStrategy = 'MERGE\_LONGEST\_BORDER');

☒☒

A window function which alters the edges of a polygonal coverage to ensure that none of the polygons overlap, that small gaps are snapped away, and that all shared edges are exactly identical. The result is a clean coverage that will pass validation tests like [ST\\_CoverageInvalidEdges](#)

The *gapMaximumWidth* controls the cleaning of gaps between polygons. Gaps smaller than this tolerance will be closed.

The *snappingDistance* controls the node snapping step, when nearby vertices are snapped together. The default setting (-1) applies an automatic snapping distance based on an analysis of the input. Set to 0.0 to turn off all snapping.

The *overlapMergeStrategy* controls the algorithm used to determine which neighboring polygons to merge overlapping areas into.

MERGE\_LONGEST\_BORDER chooses polygon with longest common border

MERGE\_MAX\_AREA chooses polygon with maximum area

MERGE\_MIN\_AREA chooses polygon with minimum area

MERGE\_MIN\_INDEX chooses polygon with smallest input index

Availability: 3.6.0 - requires GEOS >= 3.14.0

☒☒

```
-- Populate demo table
CREATE TABLE example AS SELECT * FROM (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)
) AS v(id, geom);

-- Prove it is a dirty coverage
SELECT ST_AsText(ST_CoverageInvalidEdges(geom) OVER ())
FROM example;

-- Clean the coverage
CREATE TABLE example_clean AS
  SELECT id, ST_CoverageClean(geom) OVER () AS GEOM
  FROM example;

-- Prove it is a clean coverage
SELECT ST_AsText(ST_CoverageInvalidEdges(geom) OVER ())
FROM example_clean;
```

☒☒

[ST\\_CoverageInvalidEdges](#), [ST\\_Union](#) [ST\\_CoverageSimplify](#)

## 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);

**Notes**

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.

**History:** 2.0.0 **Added**, **Added** TIN **Added**.

Availability: 1.1.2. Name changed from Affine to ST\_Affine in 1.2.2



### Note

1.3.4 **Added** (curve) **Added**. 1.3.4 **Added**.

- ✓ 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.

☒☒

```
--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)
```

☒☒

**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);
```

☒☒

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).

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒, ☒☒☒☒ TIN ☒☒☒☒☒☒☒☒☒☒.

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).

☒☒

```
--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)
```

☒☒

**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);

☒☒

Rotates a geometry geomA - rotRadians about the X axis.



**Note**

ST\_RotateX(geomA, rotRadians) is short-hand for ST\_Affine(geomA, 1, 0, 0, 0, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0).

Availability: 2.0.0. Name changed from RotateX to ST\_RotateX in 1.2.2.

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).

SQL

```
--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)
```

SQL

[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);

SQL

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

Availability: 2.0.0. Name changed from RotateY to ST\_RotateY 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).

例

```
--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)
```

例

[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);

例

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).

例: 2.0.0 例, TIN 例.

Availability: 1.1.2. Name changed from RotateZ to ST\_RotateZ in 1.2.2



#### Note

1.3.4 例 (curve) 例. 1.3.4 例.



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).

☐☐

```
--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))
```

☐☐

**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);
```

☐☐

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

1.3.4 中文文档 (curve) 中文文档. 1.3.4 中文文档.

Availability: 1.1.0.

国国国: 2.0.0 国国国国国国国国, 国国国 TIN 国国国国国国国国国国.

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.

国国

```
--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)
```

国国

**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 deltax);  
 geometry **ST\_Translate**(geometry g1, float deltax, float deltax, float deltax);

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

1.3.4 (curve) . 1.3.4 .

1.2.2 .



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



This method supports Circular Strings and Curves.

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))
```



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);
```



Translates the geometry using the deltaX and deltaY args, then scales it using the XFactor, YFactor args, working in 2D only.



Note!

### 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

1.3.4  $\frac{1}{x^2} = x^{-2}$  (curve)  $y = -\frac{1}{x}$ . 1.3.4  $\frac{1}{x^2} = x^{-2}$ .

Availability: 1.1.0.



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



This method supports Circular Strings and Curves.



```
SELECT ST_AsEWKT(ST_TransScale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 1, 1, 2));
```

```
LINESTRING(1.5 6 3,1.5 4 1)
```

```
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))
```

☒☒

[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);

☒☒

A window function that returns a cluster number for each input geometry, using the 2D [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 determine each cluster.

An input geometry is 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. Either assignment would be correct, so the border geometry will be arbitrarily assigned to one of the available clusters. In this situation it is possible for a correct cluster to be generated with fewer than minpoints geometries. To ensure deterministic assignment of border geometries (so that repeated calls to `ST_ClusterDBSCAN` will produce identical results) use an `ORDER BY` clause in the window definition. Ambiguous cluster assignments may differ from other DBSCAN implementations.



#### Note

Geometries that do not meet the criteria to join any cluster are assigned a cluster number of NULL.

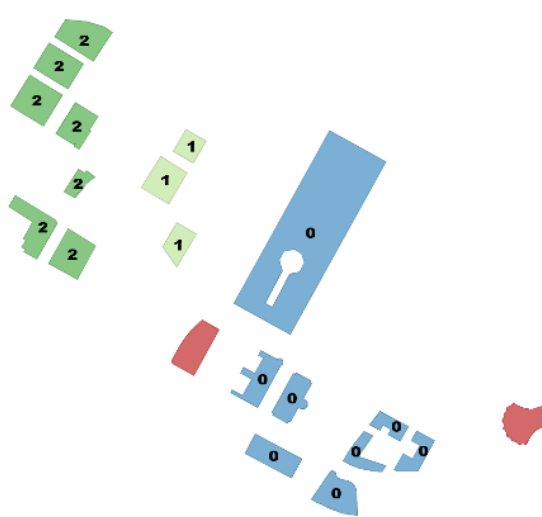
2.3.0 ☒☒☒☒☒☒☒☒☒☒.



This method supports Circular Strings and Curves.

☒☒

Clustering polygon within 50 meters of each other, and requiring at least 2 polygons per cluster.



*Clusters within 50 meters with at least 2 items 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);
```

| bucket | name                                |  |
|--------|-------------------------------------|--|
| 0      | 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             |  |
| 1      | 100 Northern Avenue                 |  |
| 1      | 100 Pier 4                          |  |
| 1      | The Institute of Contemporary Art   |  |
| 2      | 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)

A example showing combining parcels with the same cluster number into geometry collections.

```
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;
```



ST\_DWithin, ST\_ClusterKMeans, ST\_ClusterIntersecting, ST\_ClusterIntersectingWin, ST\_ClusterWithin, ST\_ClusterWithinWin



## 7.17.2 ST\_ClusterIntersecting

**ST\_ClusterIntersecting** — Aggregate function that clusters input geometries into connected sets.

### Synopsis

```
geometry[] ST_ClusterIntersecting(geometry set g);
```

ⓘ

An aggregate function that returns an array of GeometryCollections partitioning the input geometries into connected clusters that are disjoint. Each geometry in a cluster intersects at least one other geometry in the cluster, and does not intersect any geometry in other clusters.

2.2.0 ⓘ

ⓘ

```
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))
```

ⓘ

[ST\\_ClusterIntersectingWin](#), [ST\\_ClusterWithin](#), [ST\\_ClusterWithinWin](#)

## 7.17.3 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);
```

A window function that builds connected 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

```
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       |

[ST\\_ClusterIntersecting](#), [ST\\_ClusterWithin](#), [ST\\_ClusterWithinWin](#)

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 k , float8 max\_radius );

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.

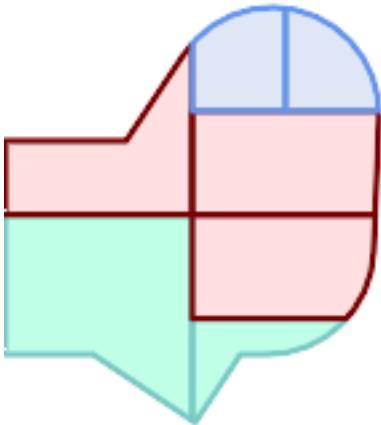
Enhanced: 3.2.0 Support for max\_radius

Enhanced: 3.1.0 Support for 3D geometries and weights

2.3.0

Generate dummy set of parcels for examples:

```
CREATE TABLE parcels AS
SELECT lpad((row_number() over())::text,3,'0') As parcel_id, geom,
('{'residential, commercial}'::text[])[1 + mod(row_number()OVER(),2)] As type
FROM
  ST_Subdivide(ST_Buffer('SRID=3857;LINESTRING(40 100, 98 100, 100 150, 60 90)'::geometry ←
    40, 'endcap=square'),12) As geom;
```



Parcels color-coded by cluster number (cid)

```
SELECT ST_ClusterKMeans(geom, 3) OVER() AS cid, parcel_id, geom
FROM parcels;
```

| cid | parcel_id | geom          |
|-----|-----------|---------------|
| 0   | 001       | 0103000000... |
| 0   | 002       | 0103000000... |
| 1   | 003       | 0103000000... |
| 0   | 004       | 0103000000... |
| 1   | 005       | 0103000000... |
| 2   | 006       | 0103000000... |
| 2   | 007       | 0103000000... |

Partitioning parcel clusters by type:

```
SELECT ST_ClusterKMeans(geom, 3) over (PARTITION BY type) AS cid, parcel_id, type
FROM parcels;
```

| cid | parcel_id | type        |
|-----|-----------|-------------|
| 1   | 005       | commercial  |
| 1   | 003       | commercial  |
| 2   | 007       | commercial  |
| 0   | 001       | commercial  |
| 1   | 004       | residential |
| 0   | 002       | residential |
| 2   | 006       | residential |

Example: Clustering a preaggregated planetary-scale data population dataset using 3D clusering and weighting. Identify at least 20 regions based on **Kontur Population Data** that do not span more than 3000 km from their center:

```
create table kontur_population_3000km_clusters as
select
  geom,
  ST_ClusterKMeans(
    ST_Force4D(
      ST_Transform(ST_Force3D(geom), 4978), -- cluster in 3D XYZ CRS
      mvalue => population -- set clustering to be weighed by population
    ),
    20, -- aim to generate at least 20 clusters
    max_radius => 3000000 -- but generate more to make each under 3000 km radius
  ) over () as cid
from
  kontur_population;
```



World population clustered to above specs produces 46 clusters. Clusters are centered at well-populated regions (New York, Moscow). Greenland is one cluster. There are island clusters that span across the antimeridian. Cluster edges follow Earth’s curvature.

**ST\_ClusterDBSCAN**, **ST\_ClusterIntersectingWin**, **ST\_ClusterWithinWin**, **ST\_ClusterIntersecting**, **ST\_ClusterST\_Subdivide**, **ST\_Force3D**, **ST\_Force4D**,

7.17.5 ST\_ClusterWithin

ST\_ClusterWithin — Aggregate function that clusters geometries by separation distance.

Synopsis

geometry[] **ST\_ClusterWithin**(geometry set g, float8 distance);

☒☒

An aggregate function that returns an array of GeometryCollections, where each collection is a cluster containing some input geometries. Clustering partitions the input geometries into sets in which each geometry is within the specified *distance* of at least one other geometry in the same cluster. Distances are Cartesian distances in the units of the SRID.

ST\_ClusterWithin is equivalent to running **ST\_ClusterDBSCAN** with `minpoints => 0`.

2.2.0 ☒☒☒☒☒☒☒☒☒☒.



This method supports Circular Strings and Curves.

☒☒

```
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))
```

☒☒

**ST\_ClusterWithinWin**, **ST\_ClusterDBSCAN**, **ST\_ClusterIntersecting**, **ST\_ClusterIntersectingWin**

### 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);

☒☒

A window function that returns a cluster number for each input geometry. Clustering partitions the geometries into sets in which each geometry is within the specified distance of at least one other geometry in the same cluster. Distances are Cartesian distances in the units of the SRID.

ST\_ClusterWithinWin is equivalent to running **ST\_ClusterDBSCAN** with `minpoints => 0`.

Availability: 3.4.0



This method supports Circular Strings and Curves.



```
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;
```

SQL

```
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)

SQL

**Box3D, ST\_GeomFromText**

## 7.18.2 Box3D

Box3D — Returns a BOX3D representing the 3D extent of a geometry.

### Synopsis

box3d **Box3D**(geometry geom);

SQL

Returns a **box3d** representing the 3D extent of the geometry.

Availability: 2.0.0 **Box3D**, **ST\_GeomFromText**, **ST\_GeomFromEWKT**, **ST\_GeomFromGeoJSON**, **ST\_GeomFromGML**, **ST\_GeomFromKML**, **ST\_GeomFromSVG**, **ST\_GeomFromTIN**, **ST\_GeomFromWKB**, **ST\_GeomFromWKT**.

- ✓ 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.

SQL

```
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)

☒☒

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);
```

☒☒

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.



#### Note

Escaping names for tables and/or namespaces that include special characters and quotes may require special handling. A user notes: "For schemas and tables, use identifier escaping rules to produce a double-quoted string, and afterwards remove the first and last double-quote character. For geometry column pass as is."

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.

☒☒



```
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)
```

☒☒

**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);
```

☒☒

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.

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒, ☒☒☒☒ TIN ☒☒☒☒☒☒☒☒☒☒.

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).

---

**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))
```

---

**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);

---

An aggregate function that returns a **box2d** bounding box that bounds a set of geometries. The bounding box coordinates are in the spatial reference system of the input geometries. **ST\_Extent** is similar in concept to Oracle Spatial/Locator's **SDO\_AGGR\_MBR**.

**Note**

**ST\_Extent** returns boxes with only X and Y ordinates even with 3D geometries. To return XYZ ordinates use **ST\_3DExtent**.

---



**Note**  
The returned box3d value does not include a SRID. Use `ST_SetSRID` to convert it into a geometry with SRID metadata. The SRID is the same as the input geometries.

2.0.0, TIN.

- ✓ This function supports Polyhedral surfaces.
- ✓ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



**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))
```

`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);

**Notes**

An aggregate function that returns a **box3d** (includes Z ordinate) bounding box that bounds a set of geometries.

The bounding box coordinates are in the spatial reference system of the input geometries.



### Note

The returned box3d value does not include a SRID. Use **ST\_SetSRID** to convert it into a geometry with SRID metadata. The SRID is the same as the input geometries.

**Changes:** 2.0.0 **Changed:** 2.0.0 In prior versions this used to be called ST\_Extent3D

**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).

**Notes**

```
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)
```

**Notes**

**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);

⌘⌘

Creates a **box2d** defined by two Point geometries. This is useful for doing range queries.

⌘⌘

```
--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)
```

⌘⌘

**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);

⌘⌘

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

⌘⌘

```
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)
```

☒☒

`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);

☒☒

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.

☒☒

```
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
```

☒☒

**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);

☒☒

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.

☒☒

```
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
```

☒☒

**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);

☒☒

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.

☒☒

```
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
```



☒☒

**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);

☒☒

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.

☒☒

```
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
```

☒☒

`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);`

☒☒

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.

☒☒

```
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
```

☒☒

`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);`

☒☒

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.

☒☒

```
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
```

文档

[ST\\_GeomFromEWKT](#), [ST\\_GeomFromText](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#)

## 7.19 文档 (Linear Referencing)

### 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);

文档

返回沿给定线串在指定比例位置上的点。比例值必须在 0 到 1 之间。  
Float8 参数指定比例位置。

该函数与 [ST\\_LineLocatePoint](#) 函数一起使用。



#### Note

This function computes points in 2D and then interpolates values for Z and M, while [ST\\_3DLineInterpolatePoint](#) computes points in 3D and only interpolates the M value.



#### Note

1.1.1 版本中，M 和 Z 值（如果有）将被插值。比例值 0.0 返回线串的第一个点。

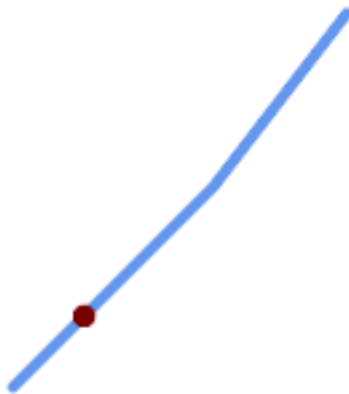
0.8.2 版本中，1.1.1 版本中 Z 和 M 值将被插值。

文档: 2.1.0 版本, 在 2.0.x 版本中，ST\_Line\_Interpolate\_Point 函数。



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

图



20% 点 (0.20) 沿线的插值点

```
-- 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)
```

图

[ST\\_LineInterpolatePoints](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineMerge](#)

## 7.19.2 ST\_3DLineInterpolatePoint

**ST\_3DLineInterpolatePoint** — Returns a point interpolated along a 3D line at a fractional location.

### Synopsis

geometry **ST\_3DLineInterpolatePoint**(geometry a\_linestring, float8 a\_fraction);



Figure 7.19.3

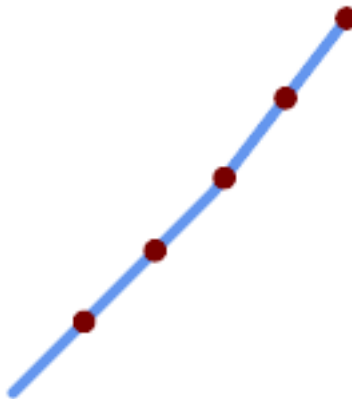


Figure 7.19.3: A `LineString` with points interpolated every 20%

```
--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))
```

Figure 7.19.4

`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);
```

Figure 7.19.4

Figure 7.19.4: The `ST_LineLocatePoint` function returns the fractional location of the closest point on a line to a point. The result is a float8 value between 0 and 1.

The `ST_LineLocatePoint` function is implemented using the `ST_LineInterpolatePoint` and `ST_LineSubstring` functions.

The `ST_LineLocatePoint` function is implemented using the `ST_LineInterpolatePoint` and `ST_LineSubstring` functions.

1.1.0 The `ST_LineLocatePoint` function was added.

Changes: 2.1.0 Added, 2.0.x The `ST_LineLocatePoint` function was added.

SQL

```
--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)
```

SQL

**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);

SQL

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.

'SQL' SQL 'SQL' ST\_LineInterpolatePoint ST\_LineInterpolatePoint



**Note**

This only works with LINESTRINGs. To use on contiguous MULTILINESTRINGs first join them with **ST\_LineMerge**.

**Note**

1.1.1 简体中文 M Z (简体中文) 简体中文。 简体中文。 简体中文。

Enhanced: 3.4.0 - Support for geography was introduced.

简体中文: 2.1.0 简体中文, 2.0.x 简体中文 ST\_Line\_Substring 简体中文。

1.1.0 简体中文。 1.1.1 简体中文 Z M 简体中文。



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

图

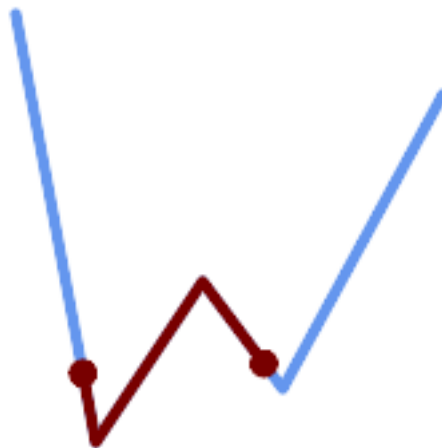


图 1/3 图 (0.333, 0.666) 简体中文

```
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)    |

Geography implementation measures along a spheroid, geometry along a line

```
SELECT ST_AsText(ST_LineSubstring( 'LINESTRING(-118.2436 34.0522, -71.0570 42.3611)::' ␣
    geography, 0.333, 0.666),6) AS geog_sub
, ST_AsText(ST_LineSubstring('LINESTRING(-118.2436 34.0522, -71.0570 42.3611)::' ␣
    geometry, 0.333, 0.666),6) AS geom_sub;
-----
geog_sub | LINESTRING(-104.167064 38.854691,-87.674646 41.849854)
geom_sub | LINESTRING(-102.530462 36.819064,-86.817324 39.585927)
```



**ST\_Length**, **ST\_LineExtend**, **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 geom\_with\_measure, float8 measure, float8 offset = 0);



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.

### 1.1.0 ST\_Locate\_Along\_Measure

Figure 2.0.0: ST\_Locate\_Along\_Measure. Returns the location of the point along the line.



This function supports M coordinates.



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.13



```
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))
```



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 geom, float8 measure_start, float8 measure_end, float8 off-
set = 0);
```



```
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))
```

￼￼

**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, float8 elevation\_start, float8 elevation\_end);

￼￼

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.

1.4.0 ￼￼￼￼￼￼￼￼￼￼.

Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE.



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

￼￼

```
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))

文档

[ST\\_Dump](#), [ST\\_LocateAlong](#), [ST\\_LocateBetween](#)

### 7.19.9 ST\_InterpolatePoint

`ST_InterpolatePoint` — 返回线性几何体在给定位置 (M 分量) 的插值点。

#### Synopsis

```
float8 ST_InterpolatePoint(geometry linear_geom_with_measure, geometry point);
```

文档

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

2.0.0 版本引入。



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

文档

```
SELECT ST_InterpolatePoint('LINESTRING M (0 0 0, 10 0 20)', 'POINT(5 5)');
-----
10
```

文档

[ST\\_AddMeasure](#), [ST\\_LocateAlong](#), [ST\\_LocateBetween](#)

### 7.19.10 ST\_AddMeasure

`ST_AddMeasure` — 在线性几何体上添加测量值。

#### Synopsis

```
geometry ST_AddMeasure(geometry geom_mline, float8 measure_start, float8 measure_end);
```



2.2.0



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

```
-- 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
```

**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);

Returns the smallest measure at which points interpolated along the given trajectories are the least distance apart.

Inputs must be valid trajectories as checked by **ST\_IsValidTrajectory**. Null is returned if the trajectories do not overlap in their M ranges.

To obtain the actual points at the computed measure use **ST\_LocateAlong**.

2.2.0



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



```
-- 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_GeometryN(ST_LocateAlong(a,m),1) pa,
         ST_GeometryN(ST_LocateAlong(b,m),1) pb
  FROM inp, cpa
)
SELECT to_timestamp(m) t,
       ST_3DDistance(pa,pb) distance,
       ST_AsText(pa, 2) AS pa, ST_AsText(pb, 2) AS pb
FROM points, cpa;
```

| t                             | distance           | pa                                     |
|-------------------------------|--------------------|----------------------------------------|
|                               | pb                 |                                        |
| 2015-05-26 10:45:31.034483-07 | 1.9652147377620688 | POINT ZM (7.59 0 3.79 1432662331.03)   |
|                               |                    | POINT ZM (9.1 1.24 3.93 1432662331.03) |

[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);`

Returns the distance (in 2D) between two trajectories at their closest point of approach. Inputs must be valid trajectories as checked by [ST\\_IsValidTrajectory](#). Null is returned if the trajectories do not overlap in their M ranges.

2.2.0 [ST\\_DistanceCPA](#).

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

❏❏

```
-- 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
-----
1.965214737762069
```

❏❏

**ST\_IsValidTrajectory**, **ST\_ClosestPointOfApproach**, **ST\_AddMeasure**, **|=|**

## 7.20.4 ST\_CPAWithin

**ST\_CPAWithin** — Tests if the closest point of approach of two trajectories is within the specified distance.

### Synopsis

boolean **ST\_CPAWithin**(geometry track1, geometry track2, float8 dist);

❏❏

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.

2.2.0 **❏❏❏❏❏❏❏❏❏❏**.



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

❏❏

```
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
```

☒☒

**ST\_IsValidTrajectory, ST\_ClosestPointOfApproach, ST\_DistanceCPA, |=|**

## 7.21 Version Functions

### 7.21.1 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);

☒☒

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`.

☒☒

```
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)
```

☒☒

Section [3.4](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Wagyu\\_Version](#), [PostGIS\\_Version](#)

### 7.21.2 PostGIS\_Full\_Version

`PostGIS_Full_Version` — Reports full PostGIS version and build configuration infos.

#### Synopsis

text `PostGIS_Full_Version()`;

☒☒

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

☒☒

```
SELECT PostGIS_Full_Version();
```

|                                                                                           | postgis_full_version |
|-------------------------------------------------------------------------------------------|----------------------|
| -----                                                                                     |                      |
| POSTGIS="3.4.0dev 3.3.0rc2-993-g61bdf43a7" [EXTENSION] PGSQL="160" GEOS="3.12.0dev-CAPI ↵ |                      |
| -1.18.0" SFCGAL="1.3.8" PROJ="7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj. ↵  |                      |
| org USER_WRITABLE_DIRECTORY=/tmp/proj DATABASE_PATH=/usr/share/proj/proj.db" GDAL="GDAL ↵ |                      |
| 3.2.2, released 2021/03/05" LIBXML="2.9.10" LIBJSON="0.15" LIBPROTOBUF="1.3.3" WAGYU ↵    |                      |
| ="0.5.0 (Internal)" TOPOLOGY RASTER                                                       |                      |
| (1 row)                                                                                   |                      |

☒☒

Section [3.4](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Wagyu\\_Version](#), [PostGIS\\_Version](#)

### 7.21.3 PostGIS\_GEOS\_Version

`PostGIS_GEOS_Version` — Returns the version number of the GEOS library.

#### Synopsis

text `PostGIS_GEOS_Version()`;

☒☒

Returns the version number of the GEOS library, or NULL if GEOS support is not enabled.

❏❏

```
SELECT PostGIS_GEOS_Version();
       postgis_geos_version
-----
3.12.0dev-CAPI-1.18.0
(1 row)
```

❏❏

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

### 7.21.4 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()`**;

❏❏

Returns the version number of the GEOS library, or against which PostGIS was built.

Availability: 3.4.0

❏❏

```
SELECT PostGIS_GEOS_Compiled_Version();
       postgis_geos_compiled_version
-----
3.12.0
(1 row)
```

❏❏

[PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Full\\_Version](#)

### 7.21.5 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()`**;

¶¶

Returns the version number of the liblwgeom library/

¶¶

```
SELECT PostGIS_Liblwgeom_Version();
postgis_liblwgeom_version
-----
3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

¶¶

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

### 7.21.6 PostGIS\_LibXML\_Version

`PostGIS_LibXML_Version` — Returns the version number of the libxml2 library.

#### Synopsis

text **`PostGIS_LibXML_Version()`**;

¶¶

Returns the version number of the LibXML2 library.

1.5 ¶¶¶¶¶¶¶¶¶¶¶¶.

¶¶

```
SELECT PostGIS_LibXML_Version();
postgis_libxml_version
-----
2.9.10
(1 row)
```

¶¶

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Version](#)

### 7.21.7 PostGIS\_LibJSON\_Version

`PostGIS_LibJSON_Version` — Returns the version number of the libjson-c library.

---

## Synopsis

text **PostGIS\_LibJSON\_Version()**;

⌘⌘

Returns the version number of the LibJSON-C library.

⌘⌘

```
SELECT PostGIS_LibJSON_Version();
 postgis_libjson_version
-----
0.17
```

⌘⌘

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Versio](#)

## 7.21.8 PostGIS\_Lib\_Build\_Date

**PostGIS\_Lib\_Build\_Date** — Returns build date of the PostGIS library.

## Synopsis

text **PostGIS\_Lib\_Build\_Date()**;

⌘⌘

Returns build date of the PostGIS library.

⌘⌘

```
SELECT PostGIS_Lib_Build_Date();
 postgis_lib_build_date
-----
2023-06-22 03:56:11
(1 row)
```

## 7.21.9 PostGIS\_Lib\_Version

**PostGIS\_Lib\_Version** — Returns the version number of the PostGIS library.

## Synopsis

text **PostGIS\_Lib\_Version()**;

---

¶¶

Returns the version number of the PostGIS library.

¶¶

```
SELECT PostGIS_Lib_Version();
 postgis_lib_version
-----
 3.4.0dev
(1 row)
```

¶¶

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

### 7.21.10 PostGIS\_PROJ\_Version

`PostGIS_PROJ_Version` — Returns the version number of the PROJ4 library.

#### Synopsis

text **`PostGIS_PROJ_Version()`**;

¶¶

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

¶¶

```
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)
```

¶¶

[PostGIS\\_PROJ\\_Compiled\\_Version](#), [PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_Version](#)

### 7.21.11 PostGIS\_PROJ\_Compiled\_Version

`PostGIS_PROJ_Compiled_Version` — Returns the version number of the PROJ library against which PostGIS was built.



## Synopsis

text **PostGIS\_PROJ\_Compiled\_Version()**;

**PostGIS\_PROJ\_Compiled\_Version()**

Returns the version number of the PROJ library, or against which PostGIS was built.

Availability: 3.5.0

**PostGIS\_PROJ\_Compiled\_Version()**

```
SELECT PostGIS_PROJ_Compiled_Version();
 postgis_proj_compiled_version
-----
 9.1.1
(1 row)
```

**PostGIS\_PROJ\_Compiled\_Version()**

**PostGIS\_PROJ\_Version**, **PostGIS\_Full\_Version**

## 7.21.12 PostGIS\_Wagyu\_Version

**PostGIS\_Wagyu\_Version** — Returns the version number of the internal Wagyu library.

## Synopsis

text **PostGIS\_Wagyu\_Version()**;

**PostGIS\_Wagyu\_Version()**

Returns the version number of the internal Wagyu library, or NULL if Wagyu support is not enabled.

**PostGIS\_Wagyu\_Version()**

```
SELECT PostGIS_Wagyu_Version();
 postgis_wagyu_version
-----
 0.5.0 (Internal)
(1 row)
```

**PostGIS\_Wagyu\_Version()**

**PostGIS\_Full\_Version**, **PostGIS\_GEOS\_Version**, **PostGIS\_PROJ\_Version**, **PostGIS\_Lib\_Version**, **PostGIS\_LibXML2\_Version**, **PostGIS\_Version**

### 7.21.13 PostGIS\_Scripts\_Build\_Date

PostGIS\_Scripts\_Build\_Date — Returns build date of the PostGIS scripts.

#### Synopsis

text **PostGIS\_Scripts\_Build\_Date**();

⌘

Returns build date of the PostGIS scripts.

1.0.0RC1 ⌘.

⌘

```
SELECT PostGIS_Scripts_Build_Date();
       postgis_scripts_build_date
-----
2023-06-22 03:56:11
(1 row)
```

⌘

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_Version](#)

### 7.21.14 PostGIS\_Scripts\_Installed

PostGIS\_Scripts\_Installed — Returns version of the PostGIS scripts installed in this database.

#### Synopsis

text **PostGIS\_Scripts\_Installed**();

⌘

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

❏❏

```
SELECT PostGIS_Scripts_Installed();
       postgis_scripts_installed
-----
3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

❏❏

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Scripts\\_Released](#), [PostGIS\\_Version](#)

### 7.21.15 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()`**;

❏❏

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

❏❏

```
SELECT PostGIS_Scripts_Released();
       postgis_scripts_released
-----
3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

❏❏

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Scripts\\_Installed](#), [PostGIS\\_Lib\\_Version](#)

### 7.21.16 PostGIS\_Version

`PostGIS_Version` — Returns PostGIS version number and compile-time options.

## Synopsis

```
text PostGIS_Version();
```



Returns PostGIS version number and compile-time options.



```
SELECT PostGIS_Version();
                                postgis_version
-----
3.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
(1 row)
```



PostGIS\_Full\_Version, PostGIS\_GEOS\_Version, PostGIS\_Lib\_Version, PostGIS\_LibXML\_Version, PostGIS\_PROJ\_Version

## 7.22 PostGIS GUC(Grand Unified Custom Variable)

### 7.22.1 postgis.gdal datapath

postgis.gdal\_datapath — GDAL 的 GDAL\_DATA 环境变量。如果存在，则  
 使用 GDAL\_DATA 环境变量。



```
GDAL ▯ GDAL_DATA ▯▯▯▯▯▯▯▯▯ PostgreSQL GUC ▯▯▯▯. postgis.gdal_datapath ▯  
▯ GDAL ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯.
```

GDAL 2.1.1 (hard-coded) 2.1.1  
GDAL 2.1.1 GDAL 2.1.1



### Note

PostgreSQL `postgresql.conf` `shared_buffers`. `shared_buffers` `shared_buffers`.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.



### Note

GDAL  GDAL DATA .

。

postgis.gdal\_datapath 。

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';
SET postgis.gdal_datapath TO default;
```

。

```
ALTER DATABASE gisdb
SET postgis.gdal_datapath = 'C:/Program Files/PostgreSQL/9.3/gdal-data';
```

。

**PostGIS\_GDAL\_Version, ST\_Transform**

## 7.22.2 postgis.gdal\_enabled\_drivers

postgis.gdal\_enabled\_drivers — PostGIS 的 GDAL 驱动。GDAL 的 GDAL\_SKIP 。

。

PostGIS 的 GDAL 驱动。GDAL 的 GDAL\_SKIP 的 PostgreSQL 的 postgresql.conf 的 。

PostgreSQL 的 POSTGIS\_GDAL\_ENABLED\_DRIVERS 的 (pass) 的 postgis.gdal\_enabled\_drivers 的 。

的 GDAL 的 的 GDAL 的 。

### Note

postgis.gdal\_enabled\_drivers 的 。



- DISABLE\_ALL 的 GDAL 的 的 DISABLE\_ALL 的 , postgis.gdal\_enabled\_drivers 的 。
- ENABLE\_ALL 的 GDAL 的 。
- VSICURL 的 GDAL 的 /vsicurl/ 的 。

postgis.gdal\_enabled\_drivers 的 DISABLE\_ALL 的 , DB 的 , ST\_FromGDALRaster(), ST\_AsGDALRaster(), ST\_AsTIFF(), ST\_AsJPEG() 的 ST\_AsPNG() 的 。



### Note

的 PostGIS 的 , postgis.gdal\_enabled\_drivers 的 DISABLE\_ALL 的 。

**Note**

GDAL\_SKIP GDAL [Configuration Options](#).

2.2.0.

To set and reset `postgis.gdal_enabled_drivers` for current session

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.gdal_enabled_drivers = default;
```

Set for all new connections to a specific database to specific drivers

```
ALTER DATABASE mygisdb SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

Setting for whole database cluster to enable all drivers. Requires super user access. Also note that database, session, and user settings override this.

```
--writes to postgres.auto.conf
ALTER SYSTEM SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
--Reloads postgres conf
SELECT pg_reload_conf();
```

[ST\\_FromGDALRaster](#), [ST\\_AsGDALRaster](#), [ST\\_AsTIFF](#), [ST\\_AsPNG](#), [ST\\_AsJPEG](#), [postgis.enable\\_outdb\\_raster](#)

### 7.22.3 `postgis.enable_outdb_rasters`

`postgis.enable_outdb_rasters` — DB

DB. PostgreSQL `postgresql.conf` (pass) `postgis.enable_outdb_rasters` `POSTGIS_ENABLE_OUTDB_RASTERS` (pass) `postgis.enable_outdb_rasters`.

PostgreSQL 0 `POSTGIS_ENABLE_OUTDB_RASTERS` (pass) `postgis.enable_outdb_rasters`.

**Note**

`postgis.enable_outdb_rasters`, GUC `postgis.enable_outdb_rasters`.

**Note**

PostGIS, `postgis.enable_outdb_rasters`.

2.2.0.



```
postgis.enable_outdb_rasters [REDACTED].
```

```
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 all new connections to a specific database

```
ALTER DATABASE gisdb SET postgis.enable_outdb_rasters = true;
```

Setting for whole database cluster. Requires super user access. Also note that database, session, and user settings override this.

```
--writes to postgres.auto.conf
ALTER SYSTEM SET postgis.enable_outdb_rasters = true;
--Reloads postgres conf
SELECT pg_reload_conf();
```



## postgis.gdal enabled drivers postgis.gdal vsi options

#### 7.22.4 postgis.gdal vsi options

`postgis.gdal_vsi_options` — DB ████████████████████████████████████████████.



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.

Availability: 3.2.0



```
postgis.enable_outdb_rasters [REDACTED].
```

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY= ↵
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy';
```

Set `postgis.gdal_vsi_options` just for the *current transaction* using the `LOCAL` keyword:

```
SET LOCAL postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxx ↵
    AWS_SECRET_ACCESS_KEY=yyyyyyyyyyyyyyyyyyyyyyyyyyyyyy';
```



```
postgis.enable_outdb_rasters postgis.gdal enabled drivers
```

### 7.22.5 postgis.gdal\_cpl\_debug

`postgis.gdal_cpl_debug` — A boolean configuration to turn logging of GDAL debug messages on and off.



By default, GDAL logging is printed to stderr, and lower level debug messages are not printed at all. Turning this GUC to true will cause GDAL logging to be sent into the PostgreSQL logging stream, so you can see more or less of it by altering the `client_min_message` PostgreSQL GUC.

Availability: 3.6.0



```
postgis.enable_outdb_rasters postgis.gdal enabled drivers
```

## 7.23 Troubleshooting Functions

### 7.23.1 PostGIS\_AddBBox

PostGIS\_AddBBox — ☒☒☒☒☒☒☒☒☒☒☒☒☒.

## Synopsis

```
geometry PostGIS_AddBBox(geometry geomA);
```

[illegible]

### Note

1. 本行在 2019 年 12 月 31 日及 2020 年 6 月 30 日，均符合《上市公司治理指引》及《上市公司章程指引》的要求。



This method supports Circular Strings and Curves.



```
UPDATE sometable
SET geom = PostGIS_AddBBox(geom)
WHERE PostGIS_HasBBox(geom) = false;
```



## PostGIS DropBBox, PostGIS HasBBox



### 7.23.2 PostGIS\_DropBBox

PostGIS\_DropBBox — XXXXXXXXXXXXXXXXXXXX.

## Synopsis

```
geometry PostGIS_DropBBox(geometry geomA);
```



ST Intersects

### Note



Note!

[illegible]

 This method supports Circular Strings and Curves.



```
--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);
```



## PostGIS AddBBox, PostGIS HasBBox, Box2D

### 7.23.3 PostGIS HasBBox

PostGIS HasBBox — `boolean`, `boolean`.



## Chapter 8

# SFCGAL Functions Reference

SFCGAL is a 2D & 3D geometry engine based on CGAL & C++ (wrapper) libraries. It is a library, not a standalone application.

SFCGAL is available at <http://www.sfcgal.org>. It is distributed as a PostgreSQL extension named `postgis_sfcgal`.

## 8.1 SFCGAL Management Functions

### 8.1.1 `postgis_sfcgal_version`

`postgis_sfcgal_version` — Returns SFCGAL version.

#### Synopsis

```
text postgis_sfcgal_version(void);
```

`text`

Returns SFCGAL version.

2.1.0 is the current version.

 This method needs SFCGAL backend.

`text`

`postgis_sfcgal_full_version`

### 8.1.2 `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\_version**(void);



Returns the full version of SFCGAL in use including CGAL and Boost versions

Availability: 3.3.0

 This method needs SFCGAL backend.



**postgis\_sfcgal\_version**

## 8.2 SFCGAL Accessors and Setters

### 8.2.1 CG\_ForceLHR

CG\_ForceLHR — LHR(Left Hand Reverse;   ).

## Synopsis

geometry **CG\_ForceLHR**(geometry geom);



Availability: 3.5.0

 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).

### 8.2.2 CG\_IsPlanar

CG\_IsPlanar —         .

## Synopsis

boolean **CG\_IsPlanar**(geometry geom);

---



## 8.2.5 CG\_Orientation

CG\_Orientation — (orientation)

### Synopsis

integer **CG\_Orientation**(geometry geom);

. -1 , 1 .

Availability: 3.5.0



This method needs SFCGAL backend.



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

## 8.2.6 CG\_Area

CG\_Area — Calculates the area of a geometry

### Synopsis

double precision **CG\_Area**( geometry geom );

Calculates the area of a geometry.

Performed by the SFCGAL module



### Note

NOTE: this function returns a double precision value representing the area.

Availability: 3.5.0



This method needs SFCGAL backend.

```
SELECT CG_Area('Polygon ((0 0, 0 5, 5 5, 5 0, 0 0), (1 1, 2 1, 2 2, 1 2, 1 1), (3 3, 4 3, 4 4, 3 4, 3 3))');
      cg_area
-----
      25
(1 row)
```

国国

**ST\_3DArea, ST\_Area**

### 8.2.7 CG\_3DArea

CG\_3DArea — 3 国国国国国国国国国国国国. 国国国国 0 国国国国国国国国.

#### Synopsis

float**CG\_3DArea**(geometry geom1);

国国

Availability: 3.5.0

- ✔ 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).

国国

国国: 国国国国 KWT 国国国国国国国国国国国国国国国国国国国国国国国国. 国国国国国国国国国国国国. 国国国国国国, 国国国国国国国国.

```
SELECT CG_3DArea(geom) As cube_surface_area,
       CG_3DArea(CG_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
```

国国

**CG\_Area, CG\_MakeSolid, CG\_IsSolid, CG\_Area**

### 8.2.8 CG\_Volume

CG\_Volume — 3 国国国国国国国国国国国国. 国国国国国国国国 (国国国国国国国国) 0 国国国国国国国国.

## Synopsis

```
float CG_Volume(geometry geom1);
```

☒☒

Availability: 3.5.0

- ✔ 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 `CG_3DVolume`)

☒☒

When closed surfaces are created with WKT, they are treated as areal rather than solid. To make them solid, you need to use `CG_MakeSolid`. Areal geometries have no volume. Here is an example to demonstrate.

```
SELECT CG_Volume(geom) As cube_surface_vol,
       CG_Volume(CG_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
```

☒☒

`CG_3DArea`, `CG_MakeSolid`, `CG_IsSolid`

## 8.2.9 ST\_ForceLHR

`ST_ForceLHR` — LHR(Left Hand Reverse; ☒☒☒☒) ☒☒☒☒☒☒☒☒.

## Synopsis

```
geometry ST_ForceLHR(geometry geom);
```





**Warning**

**ST\_ForceLHR** is deprecated as of 3.5.0. Use **CG\_ForceLHR** instead.

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).

**8.2.10 ST\_IsPlanar**

ST\_IsPlanar —

**Synopsis**

boolean **ST\_IsPlanar**(geometry geom);



**Warning**

**ST\_IsPlanar** is deprecated as of 3.5.0. Use **CG\_IsPlanar** instead.

2.2.0. 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).

**8.2.11 ST\_IsSolid**

ST\_IsSolid —

**Synopsis**

boolean **ST\_IsSolid**(geometry geom1);



## Warning

**ST\_IsSolid** is deprecated as of 3.5.0. Use **CG\_IsSolid** instead.

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).

### 8.2.12 ST\_MakeSolid

ST MakeSolid — 3D model of the part. 3D model of the part. 3D model of the part, 3D model of the part. 3D model of the part. TIN model of the part.

## Synopsis

```
geometry ST_MakeSolid(geometry geom1);
```



## Warning

**Warning**  
ST\_MakeSolid is deprecated as of 3.5.0. Use CG\_MakeSolid instead.

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).

### 8.2.13 ST Orientation

ST Orientation — ☐☐☐☐☐ (orientation) ☐☐☐☐☐.

## Synopsis

```
integer ST_Orientation(geometry geom);
```



**Warning**

**ST\_Orientation** is deprecated as of 3.5.0. Use **CG\_Orientation** instead.

2.1.0



This method needs SFCGAL backend.



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

**8.2.14 ST\_3DArea**

ST\_3DArea — 3

**Synopsis**

float**ST\_3DArea**(geometry geom1);



**Warning**

**ST\_3DArea** is deprecated as of 3.5.0. Use **CG\_3DArea** instead.

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).

: KWT



```

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

```

☒☒

[ST\\_3DArea](#), [ST\\_MakeSolid](#), [ST\\_IsSolid](#)

## 8.3 SFCGAL Processing and Relationship Functions

### 8.3.1 CG\_Intersection

CG\_Intersection — Computes the intersection of two geometries

#### Synopsis

geometry **CG\_Intersection**( geometry geomA , geometry geomB );

☒☒

Computes the intersection of two geometries.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a geometry representing the intersection.

Availability: 3.5.0



This method needs SFCGAL backend.

☒☒☒☒

```

SELECT ST_AsText(CG_Intersection('LINESTRING(0 0, 5 5)', 'LINESTRING(5 0, 0 5)'));
           cg_intersection
           -----
           POINT(2.5 2.5)
           (1 row)

```

☒☒

[ST\\_3DIntersection](#), [ST\\_Intersection](#)

### 8.3.2 CG\_Intersects

CG\_Intersects — Tests if two geometries intersect (they have at least one point in common)

#### Synopsis

boolean **CG\_Intersects**( geometry geomA , geometry geomB );

☒☒

Returns true if two geometries intersect. Geometries intersect if they have any point in common.  
Performed by the SFCGAL module



#### Note

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Availability: 3.5.0



This method needs SFCGAL backend.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒☒☒

```
SELECT CG_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
   cg_intersects
   -----
f
(1 row)
SELECT CG_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
   cg_intersects
   -----
t
(1 row)
```

☒☒

[CG\\_3DIntersects](#), [ST\\_3DIntersects](#), [ST\\_Intersects](#), [ST\\_Disjoint](#)

### 8.3.3 CG\_3DIntersects

CG\_3DIntersects — Tests if two 3D geometries intersect

## Synopsis

boolean **CG\_3DIntersects**( geometry geomA , geometry geomB );

⌘⌘

Tests if two 3D geometries intersect. 3D geometries intersect if they have any point in common in the three-dimensional space.

Performed by the SFCGAL module



### Note

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Availability: 3.5.0



This method needs SFCGAL backend.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

⌘⌘⌘⌘

```
SELECT CG_3DIntersects('POINT(1.2 0.1 0)', 'POLYHEDRALSURFACE(((0 0 0,0.5 0.5 0,1 0 0,1 1 0,0 1 0,0 0 0)),((1 0 0,2 0 0,2 1 0,1 1 0,1 0 0),(1.2 0.2 0,1.2 0.8 0,1.8 0.8 0,1.8 0.2 0,1.2 0.2 0)))');
      cg_3dintersects
      -----
      t
      (1 row)
```

⌘⌘

**CG\_Intersects**, **ST\_3DIntersects**, **ST\_Intersects**, **ST\_Disjoint**

## 8.3.4 CG\_Difference

CG\_Difference — Computes the geometric difference between two geometries

## Synopsis

geometry **CG\_Difference**( geometry geomA , geometry geomB );

XX

Computes the geometric difference between two geometries. The resulting geometry is a set of points that are present in geomA but not in geomB.

Performed by the SFCGAL module

**Note**

NOTE: this function returns a geometry.

Availability: 3.5.0



This method needs SFCGAL backend.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

XXXX

```
SELECT ST_AsText(CG_Difference('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'::geometry, 'LINESTRING ↵
(0 0, 2 2)'::geometry));
      cg_difference
      -----
POLYGON((0 0,1 0,1 1,0 1,0 0))
(1 row)
```

XX

**ST\_3DDifference, ST\_Difference**

### 8.3.5 ST\_3DDifference

ST\_3DDifference — 3 XXXXXXXXXX.

#### Synopsis

geometry **ST\_3DDifference**(geometry geom1, geometry geom2);

XX**Warning**

**ST\_3DDifference** is deprecated as of 3.5.0. Use **CG\_3DDifference** instead.



geom2 与 geom1 相交。

2.2.0 版本。

- ✔ 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).

### 8.3.6 CG\_3DDifference

CG\_3DDifference — 3 维差。

#### Synopsis

```
geometry CG_3DDifference(geometry geom1, geometry geom2);
```

返回

geom2 与 geom1 相交。

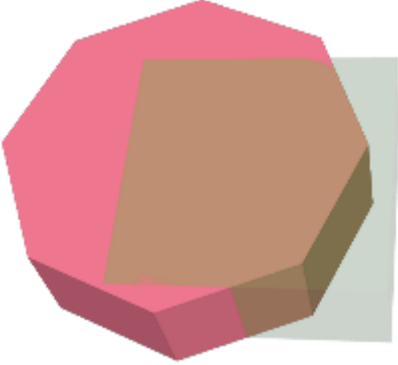
Availability: 3.5.0

- ✔ 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).

返回

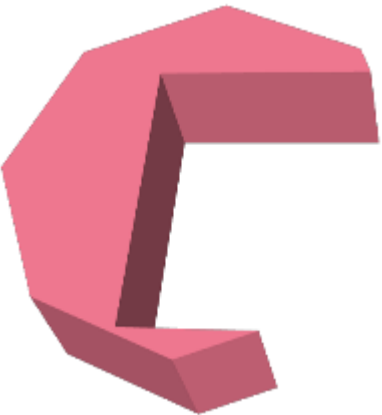
PostGIS **ST\_AsX3D** 将 3 维几何体转换为 X3Dom HTML 格式并返回 HTML 字符串。

```
SELECT CG_Extrude(ST_Buffer(
  ST_GeomFromText('POINT(100 90)'),
  50, 'quad_segs=1'),0,0,30) AS geom2;
```



3D visualization of the result of the first query. It shows a pink octagonal prism (geom1) with a brown octagonal prism (geom2) inside it, representing a buffer operation.

```
SELECT CG_3DDifference(geom1,geom2)
  AS geom1,
  CG_Extrude(
    ST_Buffer(ST_GeomFromText('POINT(80 80)'),
    50, 'quad_segs=1'),0,0,30) AS geom2 ) As t;
```



3D visualization of the result of the second query. It shows a pink octagonal prism (geom1) with a notch removed, representing a 3D difference operation.

[CG\\_Extrude](#), [ST\\_AsX3D](#), [CG\\_3DIntersection](#) [CG\\_3DUnion](#)

### 8.3.7 CG\_Distance

**CG\_Distance** — Computes the minimum distance between two geometries

#### Synopsis

double precision **CG\_Distance**( geometry geomA , geometry geomB );

Computes the minimum distance between two geometries.  
Performed by the SFCGAL module



**Note**  
NOTE: this function returns a double precision value representing the distance.

Availability: 3.5.0



This method needs SFCGAL backend.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

SQL

```
SELECT CG_Distance('LINESTRING(0.0 0.0,-1.0 -1.0)', 'LINESTRING(3.0 4.0,4.0 5.0)');
       cg_distance
-----
      2.0
(1 row)
```

See Also

[CG\\_3DDistance](#), [CG\\_Distance](#)

### 8.3.8 CG\_3DDistance

**CG\_3DDistance** — Computes the minimum 3D distance between two geometries

#### Synopsis

double precision **CG\_3DDistance**( geometry geomA , geometry geomB );

SQL

Computes the minimum 3D distance between two geometries.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a double precision value representing the 3D distance.

Availability: 3.5.0



This method needs SFCGAL backend.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

SQL

```
SELECT CG_3DDistance('LINESTRING(-1.0 0.0 2.0,1.0 0.0 3.0)', 'TRIANGLE((-4.0 0.0 1.0,4.0
0.0 1.0,0.0 4.0 1.0,-4.0 0.0 1.0))');
       cg_3ddistance
-----
      1
(1 row)
```

☒☒

**CG\_Distance**, **ST\_3DDistance**

### 8.3.9 ST\_3DConvexHull

**ST\_3DConvexHull** — ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

#### Synopsis

geometry **ST\_3DConvexHull**(geometry geom1);

☒☒



#### Warning

**ST\_3DConvexHull** is deprecated as of 3.5.0. Use **CG\_3DConvexHull** instead.

Availability: 3.3.0

- ✔ 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).

### 8.3.10 CG\_3DConvexHull

**CG\_3DConvexHull** — ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

#### Synopsis

geometry **CG\_3DConvexHull**(geometry geom1);

☒☒

Availability: 3.5.0

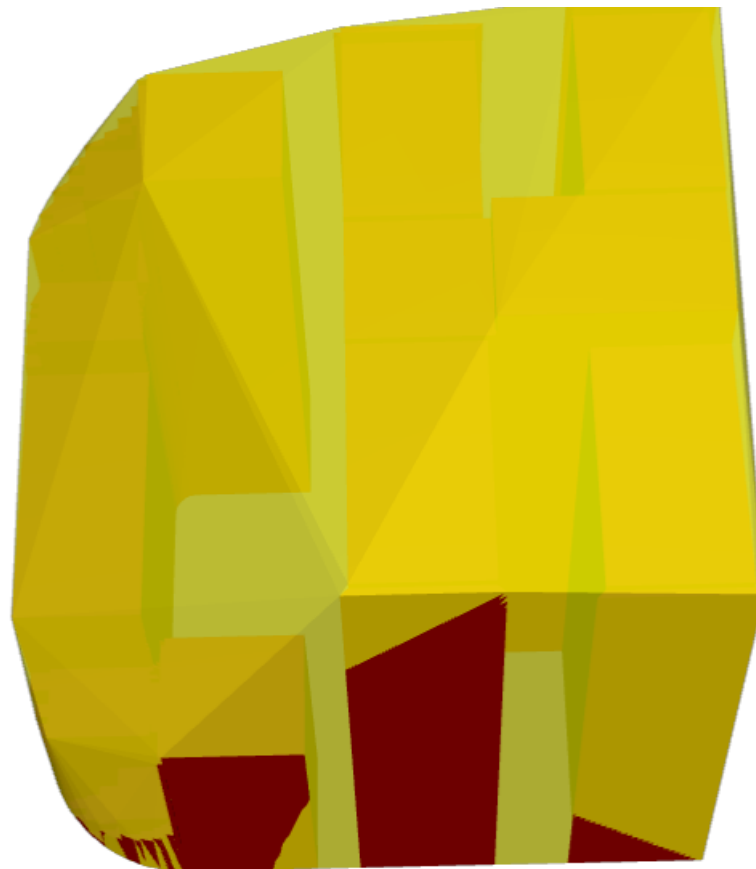
- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

❏❏

```
SELECT ST_AsText(CG_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, CG_Extrude(geom, 0,0, i ) AS geom
            FROM ST_Subdivide(ST_Letters('CH'),5) WITH ORDINALITY AS sd(geom,i)
            )
SELECT CG_3DConvexHull(ST_Collect(f.geom) )
FROM f;
```



*Original geometry overlaid with 3D convex hull*

❏❏

[ST\\_Letters](#), [ST\\_AsX3D](#)

### 8.3.11 ST\_3DIntersection

ST\_3DIntersection — 3 ❏❏❏❏❏❏❏❏❏.

## Synopsis

geometry **ST\_3DIntersection**(geometry geom1, geometry geom2);

**Warning**



### Warning

**ST\_3DIntersection** is deprecated as of 3.5.0. Use **CG\_3DIntersection** instead.

geom1  $\cap$  geom2  $\rightarrow$  geometry

2.1.0  $\rightarrow$  geometry

- ✓ 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).

## 8.3.12 CG\_3DIntersection

CG\_3DIntersection — 3  $\rightarrow$  geometry

## Synopsis

geometry **CG\_3DIntersection**(geometry geom1, geometry geom2);

**Warning**

geom1  $\cap$  geom2  $\rightarrow$  geometry

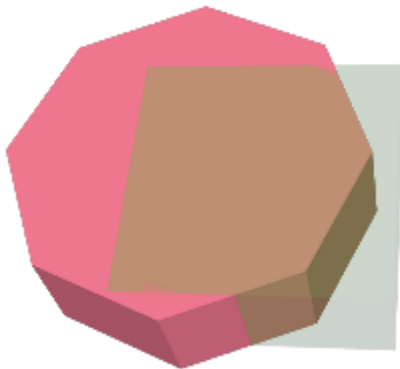
Availability: 3.5.0

- ✓ 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).

**Warning**

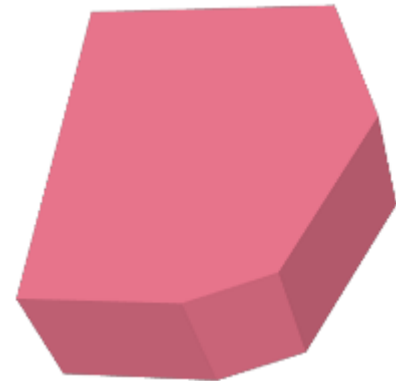
PostGIS **ST\_AsX3D**  $\rightarrow$  3  $\rightarrow$  X3Dom HTML  $\rightarrow$  HTML

```
SELECT CG_Extrude(ST_Buffer(↵
    ST_GeomFromText('POINT(100 90)'),↵
    CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'),↵
    50, '↵
    quad_segs=1'),0,0,30) AS geom2;
```



国 3 国国国国国国国国国. geom2 国国国国  
国国国国国.

```
SELECT CG_3DIntersection(geom1,geom2)↵
    50, '↵
    quad_segs=2'),0,0,30) AS geom1,↵
    quad_segs=2'),0,0,30) AS geom1,↵
    CG_Extrude(ST_Buffer(ST_GeomFromText('PO↵
    50, '↵
    quad_segs=1'),0,0,30) AS geom2 ) As t;
```



geom1 国 geom2 国国国国国

### 3 国国国国国国国国国

```
SELECT ST_AsText(CG_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)
```

国国国 (国国国国国国) 国国国 Z

```
SELECT ST_AsText(CG_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)))
```

国国国国国国国国国国国国国国国国国国国 (ST\_Dimension 国 3 国)

```
SELECT ST_AsText(CG_3DIntersection( CG_Extrude(ST_Buffer('POINT(10 20)'::geometry,10,1) ↵
,0,0,30),
CG_Extrude(ST_Buffer('POINT(10 20)'::geometry,10,1),2,0,10) ));
```

```
POLYHEDRALSURFACE Z (((13.3333333333333 13.3333333333333 10,20 20 0,20 20 ↵
10,13.3333333333333 13.3333333333333 10)),
((20 20 10,16.6666666666667 23.3333333333333 10,13.3333333333333 13.3333333333333 ↵
10,20 20 10)),
```

```
((20 20 0,16.6666666666667 23.3333333333333 10,20 20 10,20 20 0)),
((13.3333333333333 13.3333333333333 10,10 10 0,20 20 0,13.3333333333333  ←
  13.3333333333333 10)),
((16.6666666666667 23.3333333333333 10,12 28 10,13.3333333333333 13.3333333333333  ←
  10,16.6666666666667 23.3333333333333 10)),
((20 20 0,9.99999999999995 30 0,16.6666666666667 23.3333333333333 10,20 20 0)),
((10 10 0,9.99999999999995 30 0,20 20 0,10 10 0)),((13.3333333333333  ←
  13.3333333333333 10,12 12 10,10 10 0,13.3333333333333 13.3333333333333 10)),
((12 28 10,12 12 10,13.3333333333333 13.3333333333333 10,12 28 10)),
((16.6666666666667 23.3333333333333 10,9.99999999999995 30 0,12 28  ←
  10,16.6666666666667 23.3333333333333 10)),
((10 10 0,0 20 0,9.99999999999995 30 0,10 10 0)),
((12 12 10,11 11 10,10 10 0,12 12 10)),((12 28 10,11 11 10,12 12 10,12 28 10)),
((9.99999999999995 30 0,11 29 10,12 28 10,9.99999999999995 30 0)),((0 20 0,2 20  ←
  10,9.99999999999995 30 0,0 20 0)),
((10 10 0,2 20 10,0 20 0,10 10 0)),((11 11 10,2 20 10,10 10 0,11 11 10)),((12 28  ←
  10,11 29 10,11 11 10,12 28 10)),
((9.99999999999995 30 0,2 20 10,11 29 10,9.99999999999995 30 0)),((11 11 10,11 29  ←
  10,2 20 10,11 11 10)))
```

### 8.3.13 CG\_Union

CG\_Union — Computes the union of two geometries

#### Synopsis

geometry **CG\_Union**( geometry geomA , geometry geomB );



Computes the union of two geometries.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a geometry representing the union.

Availability: 3.5.0



This method needs SFCGAL backend.



```
SELECT CG_Union('POINT(.5 0)', 'LINESTRING(-1 0,1 0)');
      cg_union
      -----
LINESTRING(-1 0,0.5 0,1 0)
(1 row)
```



☒☒

`ST_3DUnion`, `ST_AsBinary`

### 8.3.14 ST\_3DUnion

`ST_3DUnion` — Perform 3D union.

#### Synopsis

geometry **ST\_3DUnion**(geometry geom1, geometry geom2);  
 geometry **ST\_3DUnion**(geometry set g1field);

☒☒



#### Warning

`ST_3DUnion` is deprecated as of 3.5.0. Use `CG_3DUnion` instead.

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.

### 8.3.15 CG\_3DUnion

`CG_3DUnion` — Perform 3D union using `postgis_sfcgal`.

#### Synopsis

geometry **CG\_3DUnion**(geometry geom1, geometry geom2);  
 geometry **CG\_3DUnion**(geometry set g1field);

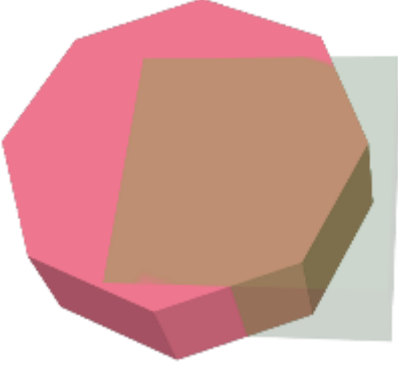
Availability: 3.5.0

- ✔ 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).

**Aggregate variant:** returns a geometry that is the 3D union of a rowset of geometries. The `CG_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.

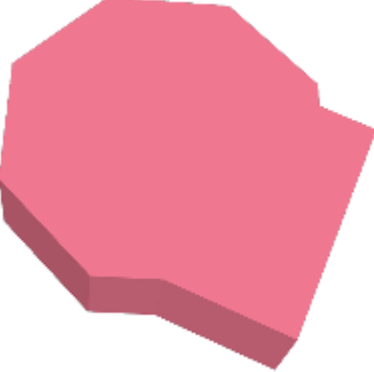
PostGIS **ST\_AsX3D** 3 **X3Dom HTML** **HTML** **HTML**.

```
SELECT CG_Extrude(ST_Buffer(
  ST_GeomFromText('POINT(100 90)'),
  CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'),
    50, 'quad_segs=1'),0,0,30) AS geom2;
```



3 `geom2`

```
SELECT CG_3DUnion(geom1,geom2)
  50, 'quad_segs=2'),0,0,30) AS geom1,
  50, 'quad_segs=2'),0,0,30) AS geom1,
  50, 'CG_Extrude(ST_Buffer(ST_GeomFrom
    50, 'quad_segs=1'),0,0,30) AS geom2 )
```



`geom1` `geom2`

**CG\_Extrude, ST\_AsX3D, CG\_3DIntersection CG\_3DDifference**

### 8.3.16 ST\_AlphaShape

ST\_AlphaShape — Computes an Alpha-shape enclosing a geometry

#### Synopsis

geometry **ST\_AlphaShape**(geometry geom, float alpha, boolean allow\_holes = false);

⊠



#### Warning

**ST\_AlphaShape** is deprecated as of 3.5.0. Use **CG\_AlphaShape** instead.

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**.

### 8.3.17 CG\_AlphaShape

CG\_AlphaShape — Computes an Alpha-shape enclosing a geometry

#### Synopsis

geometry **CG\_AlphaShape**(geometry geom, float alpha, boolean allow\_holes = false);

⊠

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.

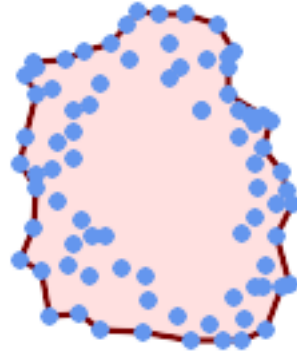
This function effectively computes a concave hull of a geometry in a similar way to **ST\_ConcaveHull**, but uses CGAL and a different algorithm.

Availability: 3.5.0 - requires SFCGAL >= 1.4.1.



This method needs SFCGAL backend.

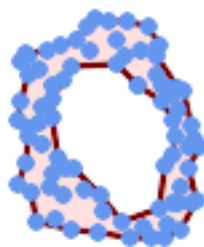
☒☒



*Alpha-shape of a MultiPoint (same example As [CG\\_OptimalAlphaShape](#))*

```
SELECT ST_AsText(CG_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));

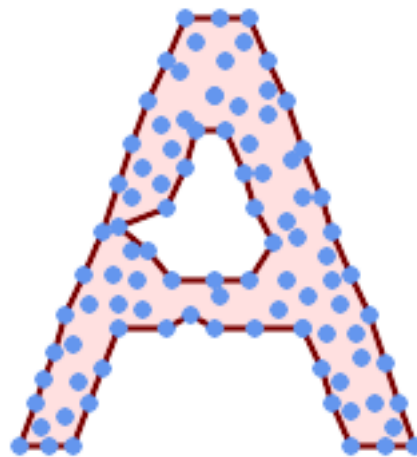
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,
37 23,30 22,28 33,23 36,26 44,27 54,23 60,24 67,27 77,
24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,
64 97,72 95,76 88,75 84,83 72,85 71,88 58,89 53))
```



*Alpha-shape of a MultiPoint, allowing holes (same example as [CG\\_OptimalAlphaShape](#))*

```
SELECT ST_AsText(CG_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 30,84 19,78 16,73 16,65 16,53 18,43 19,30 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 80,60 81,68 73,77 67,
81 60,82 54,81 47,78 43,76 27,62 22,54 32,44 42,38 46,36 61))
```



*Alpha-shape of a MultiPoint, allowing holes (same example as [ST\\_ConcaveHull](#))*

```
SELECT ST_AsText(CG_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((26 20,32 36,35 45,39 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 64,114 64,99 64,90 69,81 64,63 64,57 ←
49,52 36,46 20,37 20,26 20),
```

```
(74 93,81 109,88 124,92 138,103 138,110 122,114 109,121 96,112 82,99 82,83
82,74 93))
```

☒☒

[ST\\_ConcaveHull](#), [CG\\_OptimalAlphaShape](#)

### 8.3.18 CG\_ApproxConvexPartition

`CG_ApproxConvexPartition` — Computes approximal convex partition of the polygon geometry

#### Synopsis

geometry **CG\_ApproxConvexPartition**(geometry geom);

☒☒

Computes approximal convex partition of the polygon geometry (using a triangulation).

#### Note



A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. `CG_ApproxConvexPartition` and `CG_GreeneApproxConvexPartition` functions produce approximately optimal convex partitions. Both these functions produce convex decompositions by first decomposing the polygon into simpler polygons; `CG_ApproxConvexPartition` uses a triangulation and `CG_GreeneApproxConvexPartition` a monotone partition. These two functions both guarantee that they will produce no more than four times the optimal number of convex pieces but they differ in their runtime complexities. Though the triangulation-based approximation algorithm often results in fewer convex pieces, this is not always the case.

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.



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 `CG_StraightSkeleton` (slower case).

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).

### 8.3.20 `CG_ApproximateMedialAxis`

`CG_ApproximateMedialAxis` —

#### Synopsis

geometry **`CG_ApproximateMedialAxis`**(geometry geom);

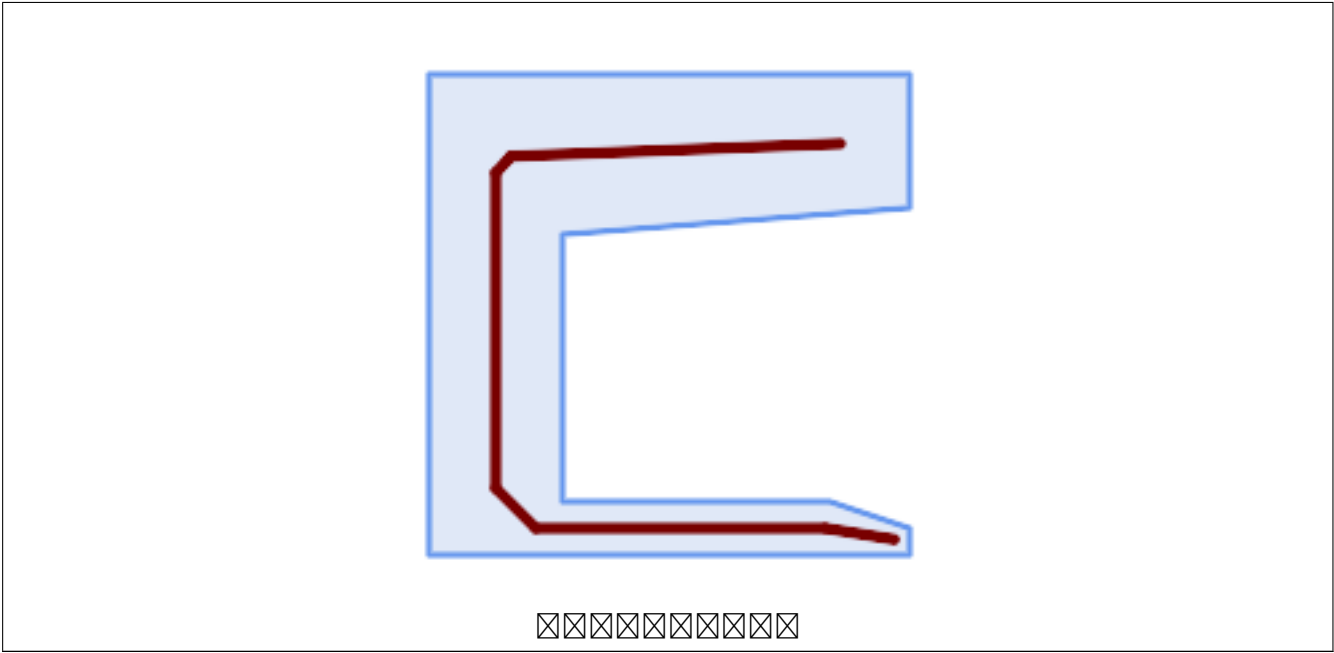
Return an approximate medial axis for the areal input based on its straight skeleton. Uses an SFCGAL specific API when built against a capable version (1.2.0+). Otherwise the function is just a wrapper around `CG_StraightSkeleton` (slower case).

Availability: 3.5.0

- ✓ 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).

```
SELECT CG_ApproximateMedialAxis(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, ↵
190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```





🔗🔗

[CG\\_StraightSkeleton](#), [CG\\_StraightSkeletonPartition](#)

### 8.3.21 ST\_ConstrainedDelaunayTriangles

`ST_ConstrainedDelaunayTriangles` — Return a constrained Delaunay triangulation around the given input geometry.

#### Synopsis

geometry **`ST_ConstrainedDelaunayTriangles`**(geometry g1);

🔗🔗



**Warning**  
`ST_ConstrainedDelaunayTriangles` is deprecated as of 3.5.0. Use `CG_ConstrainedDelaunayTriangles` instead.

Return a **Constrained Delaunay triangulation** around the vertices of the input geometry. Output is a TIN.



This method needs SFCGAL backend.

2.1.0 🔗🔗🔗🔗🔗🔗🔗🔗🔗.



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

### 8.3.22 CG\_ConstrainedDelaunayTriangles

CG\_ConstrainedDelaunayTriangles — Return a constrained Delaunay triangulation around the given input geometry.

#### Synopsis

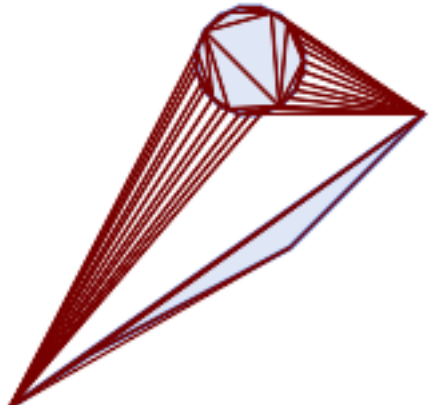
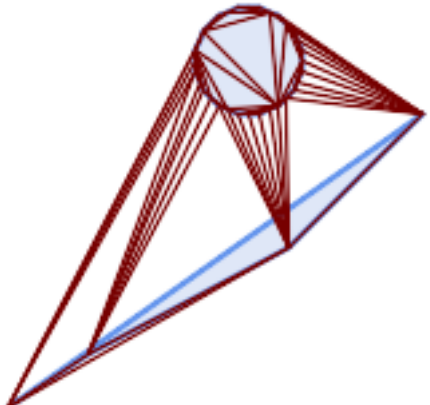
geometry **CG\_ConstrainedDelaunayTriangles**(geometry g1);

Return a **Constrained Delaunay triangulation** around the vertices of the input geometry. Output is a TIN.

✔ This method needs SFCGAL backend.

2.1.0

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

|                                                                                                                                                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  <p><i>CG_ConstrainedDelaunayTriangles of 2 polygons</i></p> <pre>select CG_ConstrainedDelaunayTriangles(<br/>  ST_Union(<br/>    POLYGON((175 150, 20 40, 50 60, 125 100, 175 150)),<br/>    ST_Buffer('POINT(110 170)::geometry', 20)<br/>  )<br/>);</pre> |  <p><i>ST_DelaunayTriangles of 2 polygons. Triangle edges cross polygon boundaries.</i></p> <pre>select ST_DelaunayTriangles(<br/>  ST_Union(<br/>    POLYGON((175 150, 20 40, 50 60, 125 100, 175 150)),<br/>    ST_Buffer('POINT(110 170)::geometry', 20)<br/>  )<br/>);</pre> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

￼￼

[ST\\_DelaunayTriangles](#), [ST\\_TriangulatePolygon](#), [CG\\_Tessellate](#), [ST\\_ConcaveHull](#), [ST\\_Dump](#)

### 8.3.23 ST\_Extrude

ST\_Extrude — ￼￼￼￼￼￼￼￼￼￼￼.

#### Synopsis

geometry **ST\_Extrude**(geometry geom, float x, float y, float z);

￼￼



#### Warning

**ST\_Extrude** is deprecated as of 3.5.0. Use **CG\_Extrude** instead.

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).

### 8.3.24 CG\_Extrude

CG\_Extrude — ￼￼￼￼￼￼￼￼￼￼￼.

#### Synopsis

geometry **CG\_Extrude**(geometry geom, float x, float y, float z);

￼￼

Availability: 3.5.0

- ✔ 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).



☒☒

**ST\_AsX3D**, **CG\_ExtrudeStraightSkeleton**

### 8.3.25 CG\_ExtrudeStraightSkeleton

CG\_ExtrudeStraightSkeleton — Straight Skeleton Extrusion

#### Synopsis

geometry **CG\_ExtrudeStraightSkeleton**(geometry geom, float roof\_height, float body\_height = 0);

☒☒

Computes an extrusion with a maximal height of the polygon geometry.

#### Note



Perhaps the first (historically) use-case of straight skeletons: given a polygonal roof, the straight skeleton directly gives the layout of each tent. If each skeleton edge is lifted from the plane a height equal to its offset distance, the resulting roof is "correct" in that water will always fall down to the contour edges (the roof's border), regardless of where it falls on the roof. The function computes this extrusion aka "roof" on a polygon. If the argument `body_height > 0`, so the polygon is extruded like with `CG_Extrude(polygon, 0, 0, body_height)`. The result is an union of these polyhedralsurfaces.

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.

☒☒

```
SELECT ST_AsText(CG_ExtrudeStraightSkeleton('POLYGON (( 0 0, 5 0, 5 5, 4 5, 4 4, 0 4, 0 0 ) ←
, (1 1, 1 2, 2 2, 2 1, 1 1))', 3.0, 2.0));
```

```
POLYHEDRALSURFACE Z (((0 0 0,0 4 0,4 4 0,4 5 0,5 5 0,5 0 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 ←
0,1 1 0)),((0 0 0,0 0 2,0 4 2,0 4 0,0 0 0)),((0 4 0,0 4 2,4 4 2,4 4 0,0 4 0)),((4 4 0,4 ←
4 2,4 5 2,4 5 0,4 4 0)),((4 5 0,4 5 2,5 5 2,5 5 0,4 5 0)),((5 5 0,5 5 2,5 0 2,5 0 0,5 5 ←
0)),((5 0 0,5 0 2,0 0 2,0 0 0,5 0 0)),((1 1 0,1 1 2,2 1 2,2 1 0,1 1 0)),((2 1 0,2 1 2,2 ←
2 2,2 2 0,2 1 0)),((2 2 0,2 2 2,1 2 2,1 2 0,2 2 0)),((1 2 0,1 2 2,1 1 2,1 1 0,1 2 0)) ←
,((0.5 2.5 2.5,0 0 2,0.5 0.5 2.5,0.5 2.5 2.5)),((1 3 3,0 4 2,0.5 2.5 2.5,1 3 3)),((0.5 ←
2.5 2.5,0 4 2,0 0 2,0.5 2.5 2.5)),((2.5 0.5 2.5,5 0 2,3.5 1.5 3.5,2.5 0.5 2.5)),((0 0 ←
2.5 0 2,2.5 0.5 2.5,0 0 2)),((0.5 0.5 2.5,0 0 2,2.5 0.5 2.5,0.5 0.5 2.5)),((4.5 3.5 ←
2.5,5 5 2,4.5 4.5 2.5,4.5 3.5 2.5)),((3.5 2.5 3.5,3.5 1.5 3.5,4.5 3.5 2.5,3.5 2.5 3.5)) ←
,((4.5 3.5 2.5,5 0 2,5 5 2,4.5 3.5 2.5)),((3.5 1.5 3.5,5 0 2,4.5 3.5 2.5,3.5 1.5 3.5)) ←
,((5 5 2,4 5 2,4.5 4.5 2.5,5 5 2)),((4.5 4.5 2.5,4 4 2,4.5 3.5 2.5,4.5 4.5 2.5)),((4.5 ←
4.5 2.5,4 5 2,4 4 2,4.5 4.5 2.5)),((3 3 3,0 4 2,1 3 3,3 3 3)),((3.5 2.5 3.5,4.5 3.5 ←
2.5,3 3 3,3.5 2.5 3.5)),((3 3 3,4 4 2,0 4 2,3 3 3)),((4.5 3.5 2.5,4 4 2,3 3 3,4.5 3.5 ←
2.5)),((2 1 2,1 1 2,0.5 0.5 2.5,2 1 2)),((2.5 0.5 2.5,2 1 2,0.5 0.5 2.5,2.5 0.5 2.5)) ←
,((1 1 2,1 2 2,0.5 2.5 2.5,1 1 2)),((0.5 0.5 2.5,1 1 2,0.5 2.5 2.5,0.5 0.5 2.5)),((1 3 ←
3,2 2 2,3 3 3,1 3 3)),((0.5 2.5 2.5,1 2 2,1 3 3,0.5 2.5 2.5)),((1 3 3,1 2 2,2 2 2,1 3 3) ←
),((2 2 2,2 1 2,2.5 0.5 2.5,2 2 2)),((3.5 2.5 3.5,3 3 3,3.5 1.5 3.5,3.5 2.5 3.5)),((3.5 ←
1.5 3.5,2 2 2,2.5 0.5 2.5,3.5 1.5 3.5)),((3 3 3,2 2 2,3.5 1.5 3.5,3 3 3)))
```

☒☒

[ST\\_Extrude](#), [CG\\_ExtrudeStraightSkeleton](#), [CG\\_StraightSkeleton](#), [CG\\_StraightSkeletonPartition](#)

### 8.3.26 CG\_GreeneApproxConvexPartition

CG\_GreeneApproxConvexPartition — Computes approximal convex partition of the polygon geometry

#### Synopsis

geometry **CG\_GreeneApproxConvexPartition**(geometry geom);

☒☒

Computes approximal monotone convex partition of the polygon geometry.

---

#### Note



A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. CG\_ApproxConvexPartition and CG\_GreeneApproxConvexPartition functions produce approximately optimal convex partitions. Both these functions produce convex decompositions by first decomposing the polygon into simpler polygons; CG\_ApproxConvexPartition uses a triangulation and CG\_GreeneApproxConvexPartition a monotone partition. These two functions both guarantee that they will produce no more than four times the optimal number of convex pieces but they differ in their runtime complexities. Though the triangulation-based approximation algorithm often results in fewer convex pieces, this is not always the case.

---

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

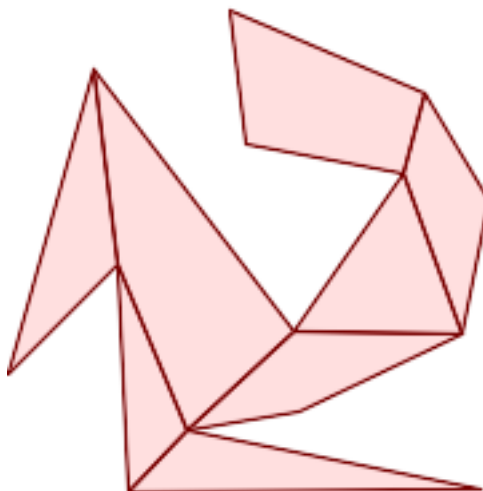
Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.

---

☒☒



*Greene Approximal Convex Partition (same example As [CG\\_YMonotonePartition](#), [CG\\_ApproxConvexPartition](#) and [CG\\_OptimalConvexPartition](#))*

```
SELECT ST_AsText(CG_GreeneApproxConvexPartition('POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150))'::geometry));
```

```
GEOMETRYCOLLECTION(POLYGON((32 159,0 45,41 86,32 159)),POLYGON((45 1,177 2,67 24,45 1)),  
POLYGON((67 24,109 31,170 60,107 61,67 24)),POLYGON((41 86,45 1,67 24,41 86)),POLYGON  
((107 61,32 159,41 86,67 24,107 61)),POLYGON((148 120,107 61,170 60,148 120)),POLYGON  
((148 120,170 60,180 110,156 150,148 120)),POLYGON((156 150,83 181,89 131,148 120,156  
150)))
```

☒☒

[CG\\_YMonotonePartition](#), [CG\\_ApproxConvexPartition](#), [CG\\_OptimalConvexPartition](#)

### 8.3.27 ST\_MinkowskiSum

ST\_MinkowskiSum — ☒☒☒☒☒☒☒☒☒☒☒☒☒.

#### Synopsis

geometry **ST\_MinkowskiSum**(geometry geom1, geometry geom2);

☒☒



#### Warning

**ST\_MinkowskiSum** is deprecated as of 3.5.0. Use [CG\\_MinkowskiSum](#) instead.

返回类型是 `geometry`, 参数, 返回类型是 `geometry` 2 个参数。

返回 A 与 B 的 Minkowski 和。A 与 B 的 Minkowski 和, 返回类型是 `geometry`。返回类型是 `geometry` (motion planning) 与 CAD(computer-aided design) 返回类型是 `geometry`。返回类型是 [Wikipedia Minkowski addition](#) 返回类型是 `geometry`。

返回类型是 `geometry` 2 个参数 (返回, 返回, 返回) 返回类型是 `geometry`。返回 3 个参数, Z 返回 0 返回类型是 `geometry` 2 个参数。返回类型是 `geometry`。返回类型是 `geometry` 2 个参数。

返回类型是 [CGAL 2D Minkowski sum](#) 返回类型是 `geometry`。

2.1.0 返回类型是 `geometry`。



This method needs SFCGAL backend.

### 8.3.28 CG\_MinkowskiSum

CG\_MinkowskiSum — 返回类型是 `geometry`。

#### Synopsis

geometry **CG\_MinkowskiSum**(geometry geom1, geometry geom2);

返回

返回类型是 `geometry`, 参数, 返回类型是 `geometry` 2 个参数。

返回 A 与 B 的 Minkowski 和。A 与 B 的 Minkowski 和, 返回类型是 `geometry`。返回类型是 `geometry` (motion planning) 与 CAD(computer-aided design) 返回类型是 `geometry`。返回类型是 [Wikipedia Minkowski addition](#) 返回类型是 `geometry`。

返回类型是 `geometry` 2 个参数 (返回, 返回, 返回) 返回类型是 `geometry`。返回 3 个参数, Z 返回 0 返回类型是 `geometry` 2 个参数。返回类型是 `geometry`。返回类型是 `geometry` 2 个参数。

返回类型是 [CGAL 2D Minkowski sum](#) 返回类型是 `geometry`。

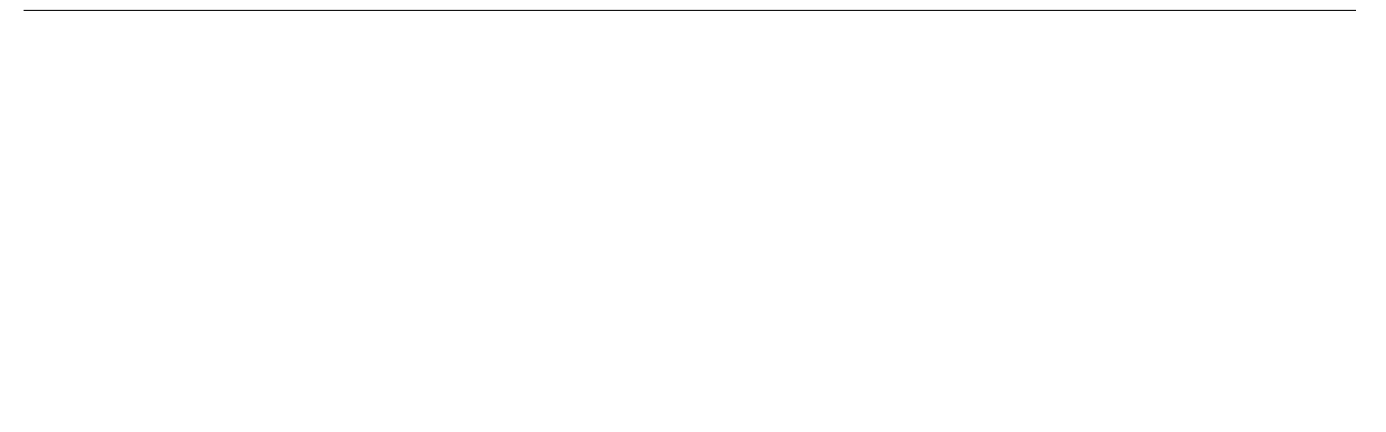
Availability: 3.5.0



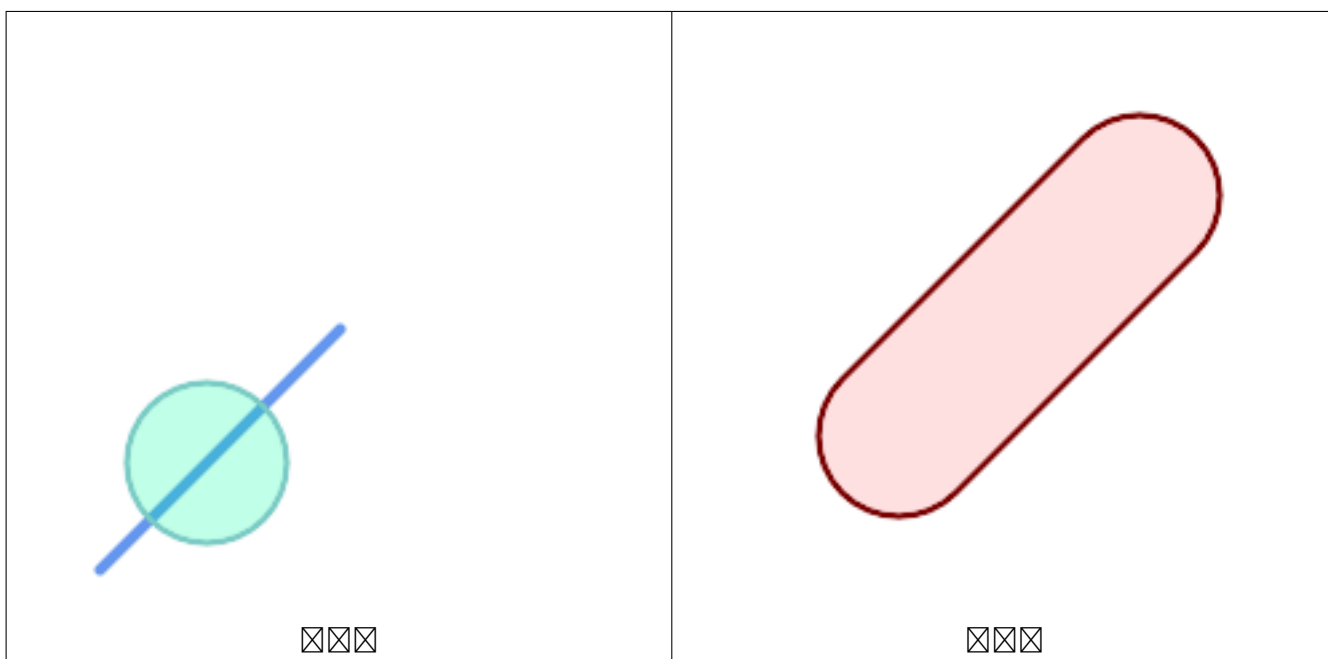
This method needs SFCGAL backend.

返回

返回类型是 `geometry`。

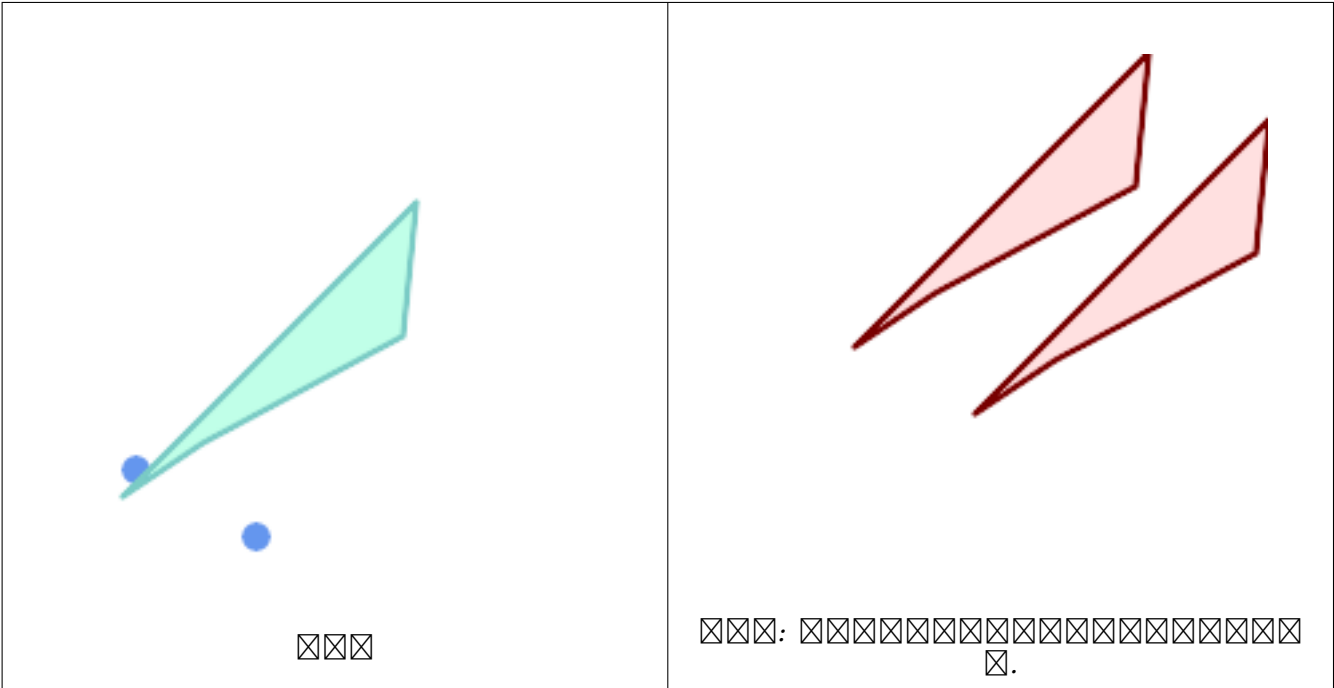






```
SELECT CG_MinkowskiSum(line, circle))
FROM (SELECT
ST_MakeLine(ST_Point(10, 10),ST_Point(100, 100)) As line,
ST_Buffer(ST_GeomFromText('POINT(50 50)'), 30) As circle) As foo;

-- 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 ↵
30.5764415879031,71.4805029709527 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)))
```



```
SELECT CG_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))
)
```

8.3.29 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);



**Warning**  
ST\_OptimalAlphaShape is deprecated as of 3.5.0. Use CG\_OptimalAlphaShape instead.

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  $\geq$  1.4.1.



This method needs SFCGAL backend.

### 8.3.30 CG\_OptimalAlphaShape

`CG_OptimalAlphaShape` — Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.

#### Synopsis

geometry **CG\_OptimalAlphaShape**(geometry geom, boolean allow\_holes = false, integer nb\_components = 1);



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

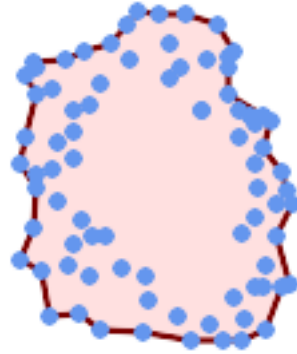
The result will not contain holes unless the optional `allow_holes` argument is specified as true.

Availability: 3.5.0 - requires SFCGAL  $\geq$  1.4.1.



This method needs SFCGAL backend.

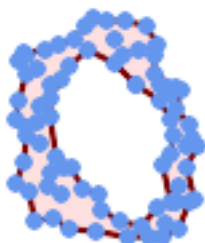
☒☒



*Optimal alpha-shape of a MultiPoint (same example as [CG\\_AlphaShape](#))*

```
SELECT ST_AsText(CG_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 [CG\\_AlphaShape](#))*

```
SELECT ST_AsText(CG_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))
```



[ST\\_ConcaveHull](#), [CG\\_AlphaShape](#)

### 8.3.31 CG\_OptimalConvexPartition

**CG\_OptimalConvexPartition** — Computes an optimal convex partition of the polygon geometry

#### Synopsis

geometry **CG\_OptimalConvexPartition**(geometry geom);



Computes an optimal convex partition of the polygon geometry.

**Note**

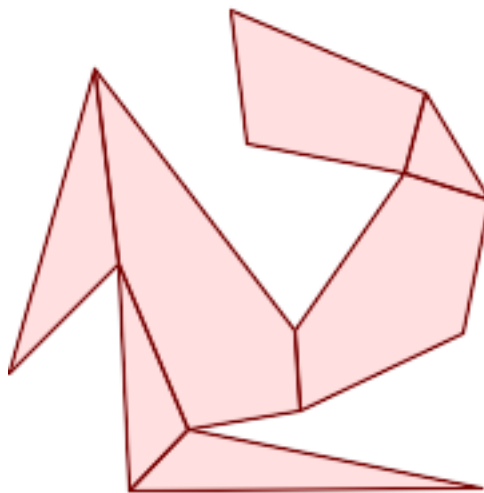
A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. `CG_OptimalConvexPartition` produces a partition that is optimal in the number of pieces.

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.



*Optimal Convex Partition (same example As [CG\\_YMonotonePartition](#), [CG\\_ApproxConvexPartition](#) and [CG\\_GreeneApproxConvexPartition](#))*

```
SELECT ST_AsText(CG_OptimalConvexPartition('POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150))'::geometry));
```

```
GEOMETRYCOLLECTION(POLYGON((156 150,83 181,89 131,148 120,156 150)),POLYGON((32 159,0 45,41 86,32 159)),POLYGON((45 1,177 2,67 24,45 1)),POLYGON((41 86,45 1,67 24,41 86)),POLYGON((107 61,32 159,41 86,67 24,109 31,107 61)),POLYGON((148 120,107 61,109 31,170 60,180 110,148 120)),POLYGON((156 150,148 120,180 110,156 150)))
```



[CG\\_YMonotonePartition](#), [CG\\_ApproxConvexPartition](#), [CG\\_GreeneApproxConvexPartition](#)

### 8.3.32 CG\_StraightSkeleton

`CG_StraightSkeleton` — (straight skeleton) .

#### Synopsis

geometry **CG\_StraightSkeleton**(geometry geom, boolean use\_distance\_as\_m = false);

☒☒

Availability: 3.5.0  
Requires SFCGAL >= 1.3.8 for option use\_distance\_as\_m

- ✔ 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).

☒☒

```
SELECT CG_StraightSkeleton(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, 190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```

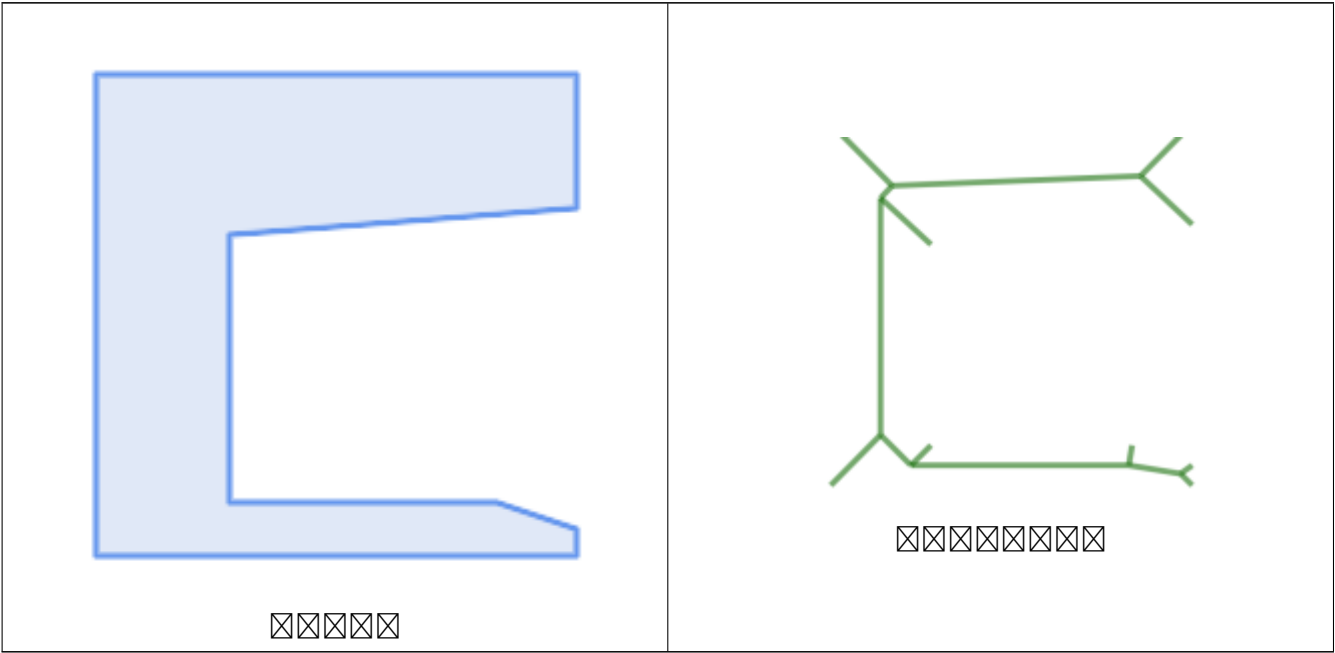
```
SELECT ST_AsText(CG_StraightSkeleton('POLYGON((0 0,1 0,1 1,0 1,0 0))', true);
```

```
MULTILINESTRING M ((0 0 0,0.5 0.5 0.5),(1 0 0,0.5 0.5 0.5),(1 1 0,0.5 0.5 0.5),(0 1 0,0.5 0.5 0.5))
```

Note that valid inputs with rings that touch at a single point will raise an error.

```
SELECT CG_StraightSkeleton('POLYGON((0 0, 3 0, 3 3, 0 3, 0 0), (0 0, 1 2, 2 1, 0 0))');
```

```
NOTICE: During straight_skeleton(A) :
NOTICE:   with A: POLYGON((0/1 0/1,3/1 0/1,3/1 3/1,0/1 3/1,0/1 0/1),(0/1 0/1,1/1 2/1,2/1 1/1,0/1 0/1))
ERROR:  straight skeleton of Polygon with point touching rings is not implemented.
```



文档

[CG\\_StraightSkeletonPartition](#), [CG\\_ExtrudeStraightSkeleton](#)

### 8.3.33 ST\_StraightSkeleton

ST\_StraightSkeleton — 计算 (straight skeleton) 骨架。

#### Synopsis

geometry **ST\_StraightSkeleton**(geometry geom);

文档



#### Warning

[ST\\_StraightSkeleton](#) is deprecated as of 3.5.0. Use [CG\\_StraightSkeleton](#) instead.

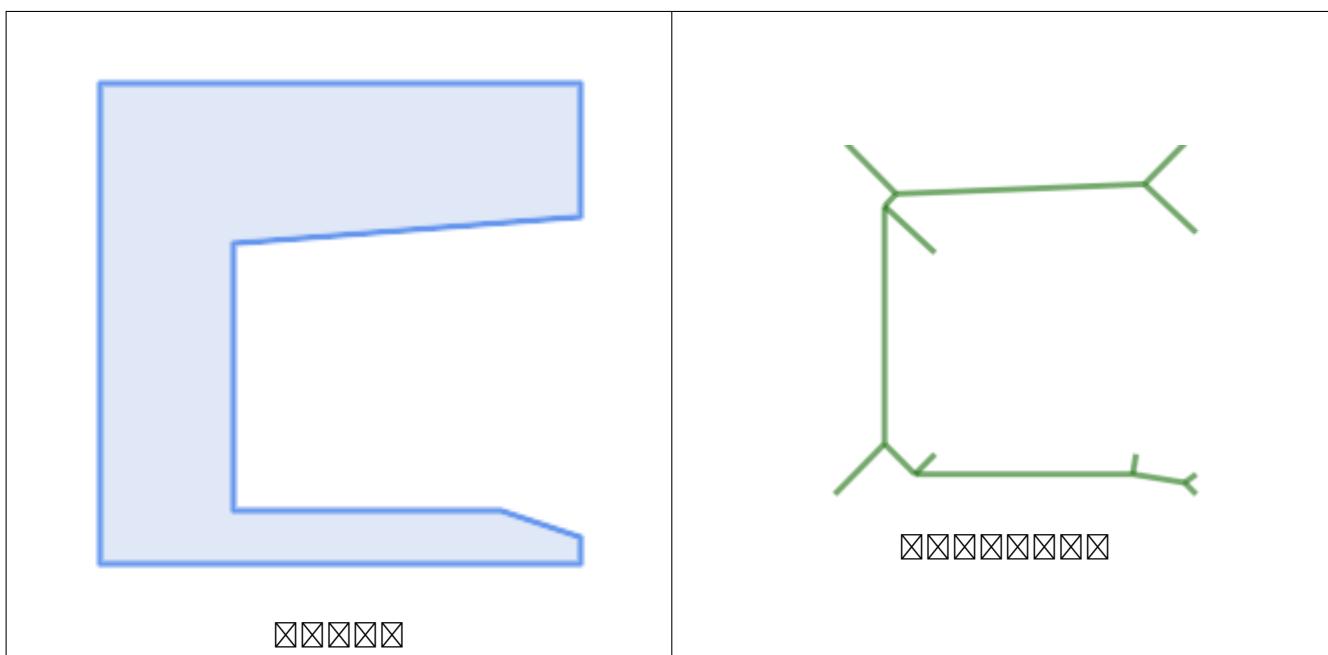
2.1.0 版本引入。

- ✓ 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).

文档

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





CG ExtrudeStraightSkeleton

### 8.3.34 ST\_Tessellate

ST Tesselate —  $\Delta \Delta \Delta \Delta \Delta \Delta \Delta \Delta \Delta \Delta \Delta \Delta \Delta \Delta \Delta$  (tessellation)  $\Delta \Delta \Delta \Delta \Delta \Delta \Delta \Delta$  TIN  $\Delta \Delta$  TIN  $\Delta \Delta \Delta \Delta$   
 $\Delta \Delta \Delta \Delta \Delta \Delta$ .

## Synopsis

```
geometry ST_Tessellate(geometry geom);
```



## Warning

**Warning**  
ST\_Tessellate is deprecated as of 3.5.0. Use CG\_Tessellate instead.

[REDACTED] (REDACTED) TIN [REDACTED]  
[REDACTED].



### Note

**ST\_TriangulatePolygon** does similar to this function except that it returns a geometry collection of polygons instead of a TIN and also only works with 2D geometries.

2.1.0 添加。

- ✔ 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).

### 8.3.35 CG\_Tessellate

`CG_Tessellate` — 三角化 (tessellation) 将 TIN 转换为 TIN 格式。

#### Synopsis

geometry **CG\_Tessellate**(geometry geom);

返回

[geometry] 三角化 (tessellation) 将 TIN 转换为 TIN 格式。



#### Note

**ST\_TriangulatePolygon** does similar to this function except that it returns a geometry collection of polygons instead of a TIN and also only works with 2D geometries.

Availability: 3.5.0

- ✔ 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).

返回

```
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 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
```



繁體中文

```
SELECT CG_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
      ((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
```

ST\_AsText 繁體中文:

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



繁體中文



**Note**

NOTE: this function returns a geometry representing the triangulated result.

Availability: 3.5.0



This method needs SFCGAL backend.

`SELECT`

```
SELECT CG_Triangulate('POLYGON((0.0 0.0,1.0 0.0,1.0 1.0,0.0 1.0,0.0 0.0),(0.2 0.2,0.2 0.8,0.8 0.8,0.8 0.2,0.2 0.2))');
      cg_triangulate
      -----
      TIN(((0.8 0.2,0.2 0.2,1 0,0.8 0.2)),((0.2 0.2,0 0,1 0,0.2 0.2)),((1 1,0.8 0.8,0.8 0.2,1 1)),((0 1,0 0,0.2 0.2,0 1)),((0 1,0.2 0.8,1 1,0 1)),((0 1,0.2 0.2,0.2 0.8,0 1)),((0.2 0.8,0.8 0.8,1 1,0.2 0.8)),((0.2 0.8,0.2 0.2,0.8 0.8)),((1 1,0.8 0.2,1 0,1 1)),((0.8 0.8,0.2 0.8,0.8 0.2,0.8 0.8)))
      (1 row)
```

`CG_ConstrainedDelaunayTriangles`, `ST_DelaunayTriangles`, `ST_TriangulatePolygon`

### 8.3.37 CG\_Visibility

`CG_Visibility` — Compute a visibility polygon from a point or a segment in a polygon geometry

#### Synopsis

```
geometry CG_Visibility(geometry polygon, geometry point);
geometry CG_Visibility(geometry polygon, geometry pointA, geometry pointB);
```

`Availability`

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.

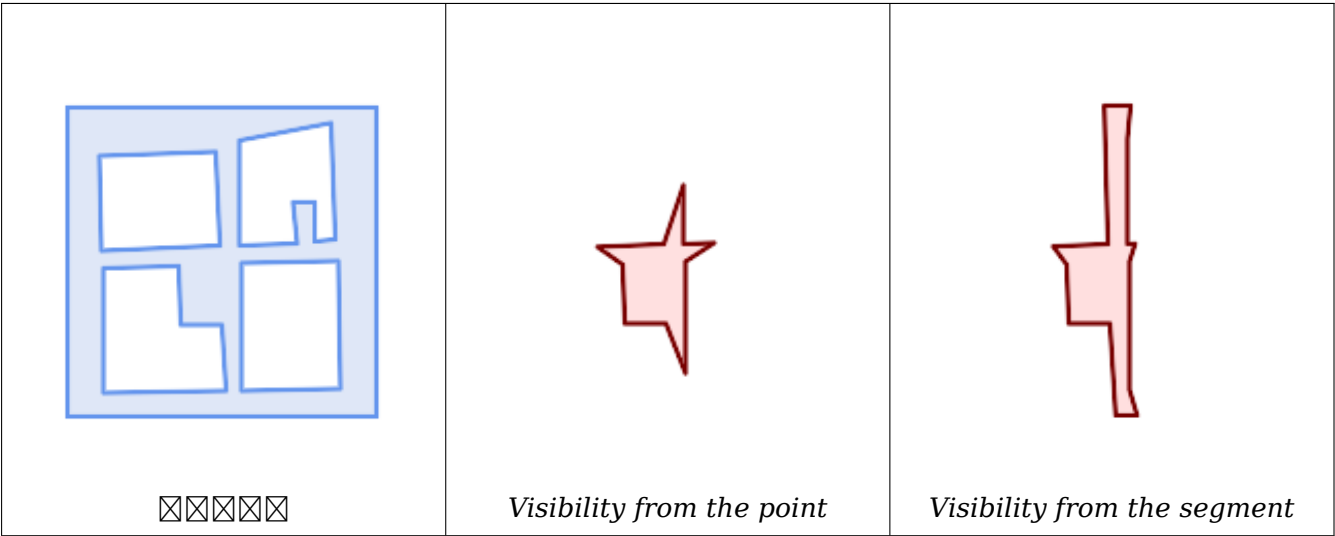


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒

```
SELECT CG_Visibility('POLYGON((23.5 23.5,23.5 173.5,173.5 173.5,173.5 23.5,23.5 23.5),(108 98,108 36,156 37,155 99,108 98),(107 157.5,107 106.5,135 107.5,133 127.5,143.5 127.5,143.5 108.5,153.5 109.5,151.5 166,107 157.5),(41 95.5,41 35,100.5 36,98.5 68,78.5 68,77.5 96.5,41 95.5),(39 150,40 104,97.5 106.5,95.5 152,39 150))'::geometry, 'POINT(91 87)'::geometry);

SELECT CG_Visibility('POLYGON((23.5 23.5,23.5 173.5,173.5 173.5,173.5 23.5,23.5 23.5),(108 98,108 36,156 37,155 99,108 98),(107 157.5,107 106.5,135 107.5,133 127.5,143.5 127.5,143.5 108.5,153.5 109.5,151.5 166,107 157.5),(41 95.5,41 35,100.5 36,98.5 68,78.5 68,77.5 96.5,41 95.5),(39 150,40 104,97.5 106.5,95.5 152,39 150))'::geometry, 'POINT(78.5 68)'::geometry, 'POINT(98.5 68)'::geometry);
```



8.3.38 CG\_YMonotonePartition

CG\_YMonotonePartition — Computes y-monotone partition of the polygon geometry

Synopsis

```
geometry CG_YMonotonePartition(geometry geom);
```

☒☒

Computes y-monotone partition of the polygon geometry.

Note

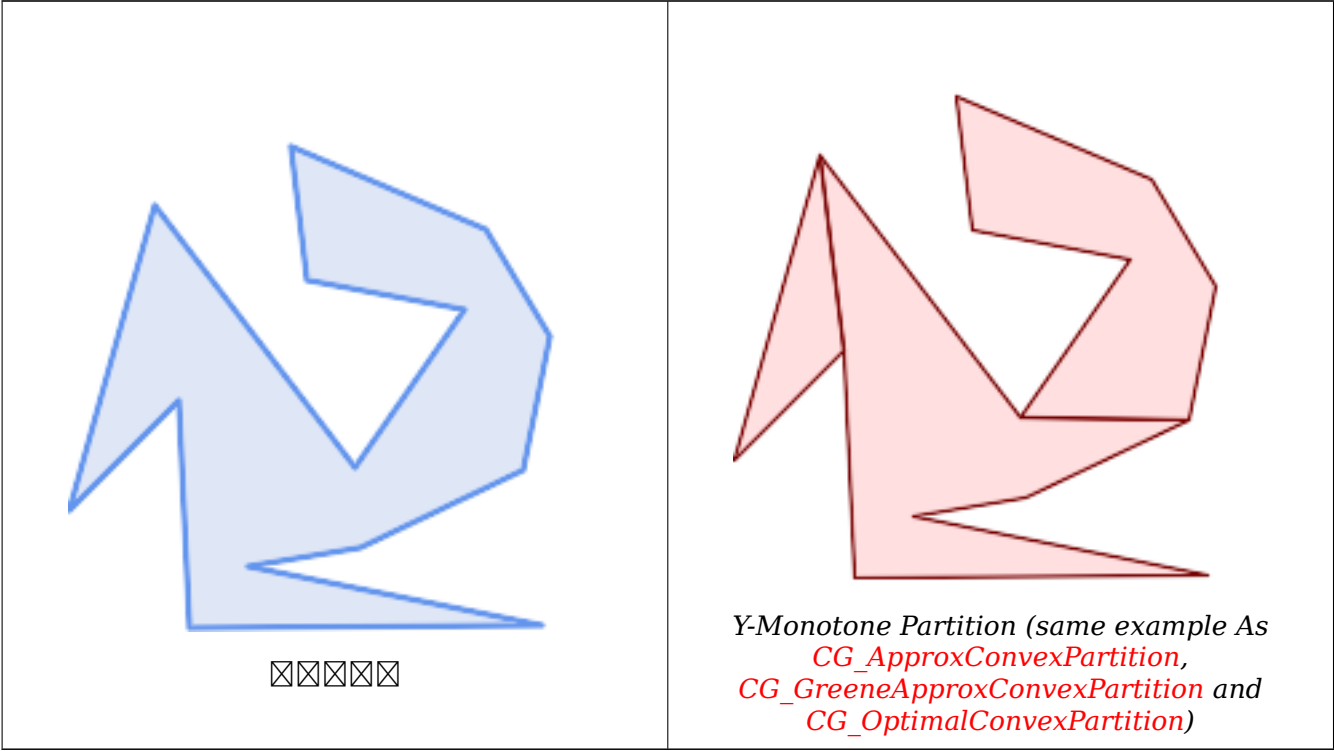
A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. A y-monotone polygon is a polygon whose vertices  $v_1, \dots, v_n$  can be divided into two chains  $v_1, \dots, v_k$  and  $v_k, \dots, v_n, v_1$ , such that any horizontal line intersects either chain at most once. This algorithm does not guarantee a bound on the number of polygons produced with respect to the optimal number.

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0

 This method needs SFCGAL backend.





```
SELECT ST_AsText(CG_YMonotonePartition('POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150))'::geometry));
```

```
GEOMETRYCOLLECTION(POLYGON((32 159,0 45,41 86,32 159)),POLYGON((107 61,32 159,41 86,45 1,177 2,67 24,109 31,170 60,107 61)),POLYGON((156 150,83 181,89 131,148 120,107 61,170 60,180 110,156 150)))
```



*CG\_ApproxConvexPartition*, *CG\_GreeneApproxConvexPartition*, *CG\_OptimalConvexPartition*

### 8.3.39 CG\_StraightSkeletonPartition

CG\_StraightSkeletonPartition — Computes the straight skeleton partition of a polygon.

#### Synopsis

geometry **CG\_StraightSkeletonPartition**(geometry geom, boolean auto\_orientation);

☒☒

Computes the straight skeleton partition of the input polygon geometry *geom*. The straight skeleton is a partitioning of the polygon into faces formed by tracing the collapse of its edges. If *auto\_orientation* is set to true, the function will automatically adjust the orientation of the input polygon to ensure correct results.

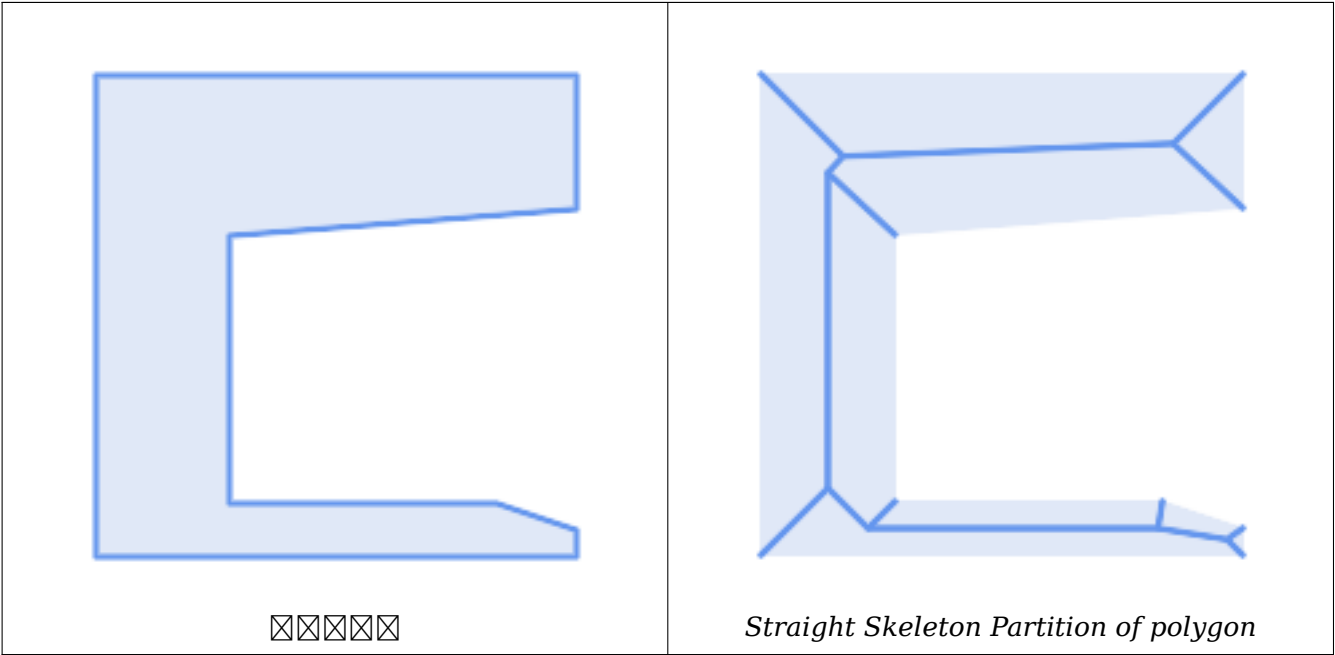
Availability: 3.6.0 - requires SFCGAL >= 2.0.0.

✔ This method needs SFCGAL backend.

☒☒

```
SELECT ST_AsText(CG_StraightSkeletonPartition('POLYGON((0 0, 4 0, 2 2, 0 0))', true));
-- Result: MULTIPOLYGON(((0 0,2 0.83,2 2)),((4 0,2 0.83,0 0)),((2 2,2 0.83,4 0)))

SELECT CG_StraightSkeletonPartition(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, 190 20
, 160 30, 60 30, 60 130, 190 140, 190 190 ))')
, true );
```



☒☒

CG\_StraightSkeleton, ST\_Polygonize

8.3.40 CG\_3DBuffer

CG\_3DBuffer — Computes a 3D buffer around a geometry.

Synopsis

geometry **CG\_3DBuffer**(geometry geom, float8 radius, integer segments, integer buffer\_type);





Generates a 3D buffer around the input geometry *geom* with a specified *radius*. The buffer is constructed in 3D space, creating a volumetric representation of the geometry's surroundings. The *segments* parameter defines the number of segments used to approximate the curved sections of the buffer, with a minimum value of 4 segments required. The *buffer\_type* specifies the type of buffer to create: 0: Rounded buffer (default) 1: Flat buffer 2: Square buffer

Input geometry must be a Point or LineString.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



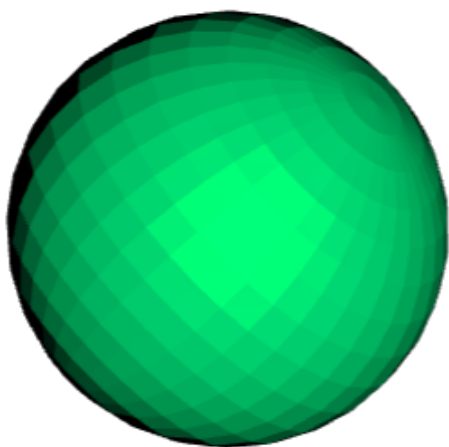
This method needs SFCGAL backend.



```
SELECT ST_AsText(CG_3DBuffer('POINT(0 0 0)', 1, 8, 0));
-- Result: POLYHEDRALSURFACE Z (((0 0 1, 0.5 -0.5 0.71, 0 -0.71 0.71, 0 0 1)), ... )
```

The following images were rendered pasting the output of the ST\_AsX3D query into [X3D Viewer](#).

```
SELECT string_agg('<Shape
>' || ST_AsX3D(cgbuffer3d_output) || '<Appearance>
    <Material diffuseColor="0 0.8 0.2" specularColor="0 1 0"/>
    </Appearance>
</Shape
>', '');
```



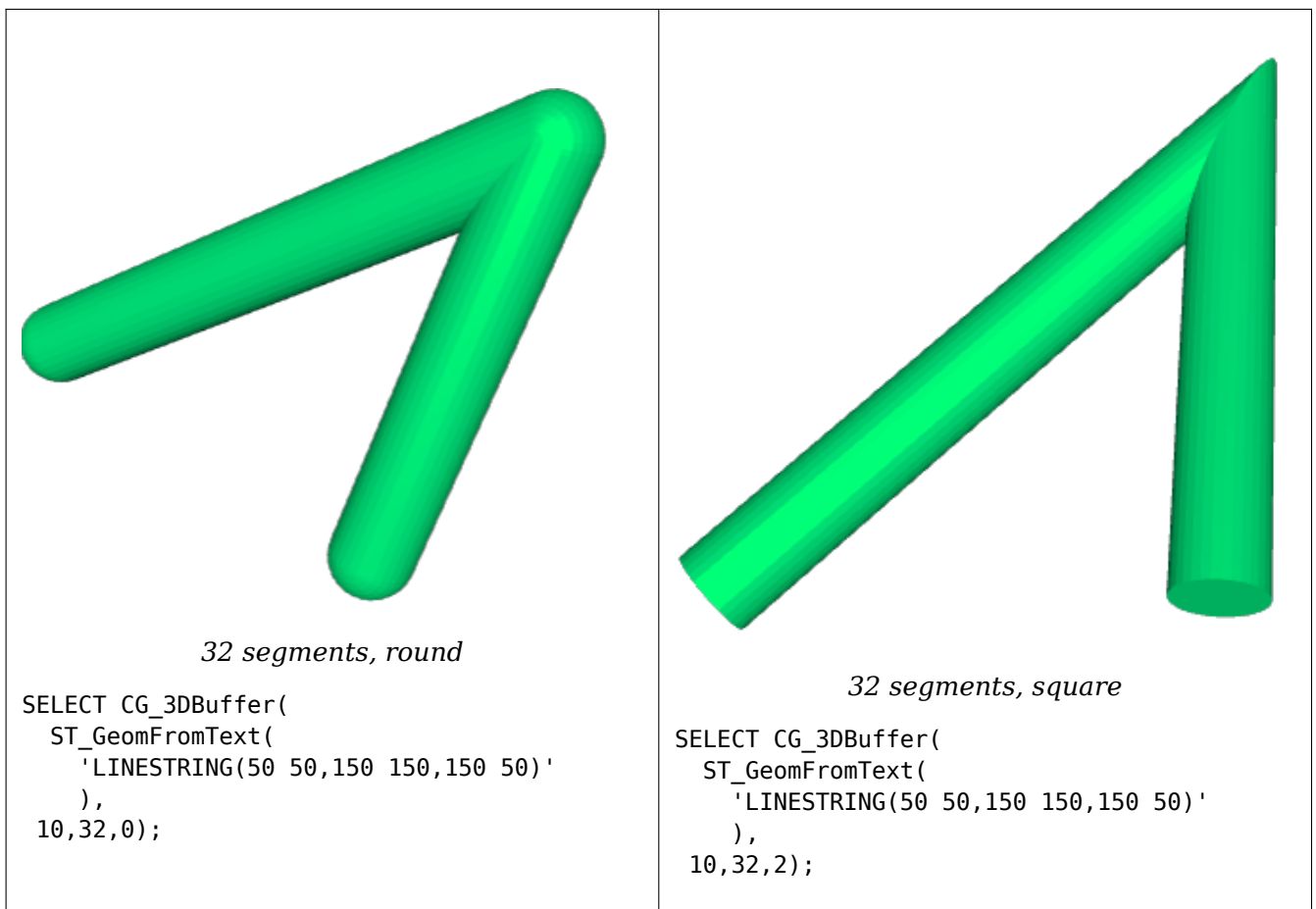
*segments=32 (rounded buffer)*

```
SELECT CG_3DBuffer(ST_GeomFromText('POINT (100 90)'), 50,32,0);
```



*5 segments rounded*

```
SELECT CG_3DBuffer(
  ST_GeomFromText('POINT(100 90)'),
  50,5,0);
```



[ST\\_Buffer](#), [ST\\_3DConvexHull](#), [ST\\_AsX3D](#)

### 8.3.41 CG\_Rotate

**CG\_Rotate** — Rotates a geometry by a given angle around the origin (0,0).

#### Synopsis

geometry **CG\_Rotate**(geometry geom, float8 angle);



Rotates the input geometry *geom* by *angle* radians around the origin point (0,0). The rotation is performed in 2D space; Z coordinates are not modified. Positive angles rotate the geometry counter-clockwise.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.

❏❏

```
SELECT ST_AsText(CG_Rotate('LINESTRING(1 0, 0 1)', pi()/2));
-- Result: LINESTRING(0 1, -1 0)
```

❏❏

[CG\\_2DRotate](#), [ST\\_Rotate](#)

### 8.3.42 CG\_2DRotate

**CG\_2DRotate** — Rotates a geometry by a given angle around a specified point in 2D.

#### Synopsis

geometry **CG\_2DRotate**(geometry geom, float8 angle, float8 cx, float8 cy);

❏❏

Rotates the input geometry *geom* by *angle* radians around the point (*cx*, *cy*). The rotation is performed in 2D space; Z coordinates are dropped. Positive angles rotate the geometry counter-clockwise.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.

❏❏

```
SELECT ST_AsText(CG_2DRotate('POINT(1 0)', pi()/2, 1, 1));
-- Result: POINT(2 1)
```

❏❏

[CG\\_Rotate](#), [CG\\_3DRotate](#)

### 8.3.43 CG\_3DRotate

**CG\_3DRotate** — Rotates a geometry in 3D space around an axis vector.

#### Synopsis

geometry **CG\_3DRotate**(geometry geom, float8 angle, float8 ax, float8 ay, float8 az);

☒☒

Rotates the input geometry *geom* by *angle* radians around an axis defined by the vector (*ax*, *ay*, *az*) passing through the origin (0,0,0).

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.



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

☒☒

```
SELECT ST_AsText(CG_3DRotate('POINT(1 0 0)', pi()/2, 0, 0, 1));
-- Result: POINT(0 1 0)
```

☒☒

**CG\_RotateX**, **CG\_RotateY**, **CG\_RotateZ**

### 8.3.44 CG\_RotateX

**CG\_RotateX** — Rotates a geometry around the X-axis by a given angle.

#### Synopsis

geometry **CG\_RotateX**(geometry geom, float8 angle);

☒☒

Rotates the input geometry *geom* by *angle* radians around the X-axis.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.



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

☒☒

```
SELECT ST_AsText(CG_RotateX('POINT(0 1 0)', pi()/2));
-- Result: POINT(0 0 1)
```

☒☒

**CG\_RotateY**, **CG\_RotateZ**, **CG\_3DRotate**

### 8.3.45 CG\_RotateY

CG\_RotateY — Rotates a geometry around the Y-axis by a given angle.

#### Synopsis

geometry **CG\_RotateY**(geometry geom, float8 angle);

⊠

Rotates the input geometry *geom* by *angle* radians around the Y-axis passing.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.



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

⊠

```
SELECT ST_AsText(CG_RotateY('POINT(1 0 0)', pi()/2));
-- Result: POINT(0 0 -1)
```

⊠

**CG\_RotateX**, **CG\_RotateZ**, **CG\_3DRotate**

### 8.3.46 CG\_RotateZ

CG\_RotateZ — Rotates a geometry around the Z-axis by a given angle.

#### Synopsis

geometry **CG\_RotateZ**(geometry geom, float8 angle);

⊠

Rotates the input geometry *geom* by *angle* radians around the Z-axis.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.



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

⊠

```
SELECT ST_AsText(CG_RotateZ('POINT(1 0 0)', pi()/2));
-- Result: POINT(0 1 0)
```

☒☒

[CG\\_RotateX](#), [CG\\_RotateY](#), [CG\\_3DRotate](#)

### 8.3.47 CG\_Scale

**CG\_Scale** — Scales a geometry uniformly in all dimensions by a given factor.

#### Synopsis

geometry **CG\_Scale**(geometry geom, float8 factor);

☒☒

Scales the input geometry *geom* by a uniform scale *factor* in all dimensions (X, Y, and Z). The scaling is performed relative to the origin point (0,0,0).

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.

☒☒

```
SELECT ST_AsText(CG_Scale('LINESTRING(1 1, 2 2)', 2));
-- Result: LINESTRING(2 2, 4 4)
```

☒☒

[CG\\_3DScale](#), [CG\\_3DScaleAroundCenter](#), [ST\\_Scale](#)

### 8.3.48 CG\_3DScale

**CG\_3DScale** — Scales a geometry by separate factors along X, Y, and Z axes.

#### Synopsis

geometry **CG\_3DScale**(geometry geom, float8 factorX, float8 factorY, float8 factorZ);

☒☒

Scales the input geometry *geom* by different factors along the X, Y, and Z axes. The scaling is performed relative to the origin point (0,0,0).

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.



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

SQL

```
SELECT ST_AsText(CG_3DScale('POINT(1 1 1)', 2, 3, 4));
-- Result: POINT(2 3 4)
```

SQL

[CG\\_Scale](#), [CG\\_3DScaleAroundCenter](#)

### 8.3.49 CG\_3DScaleAroundCenter

**CG\_3DScaleAroundCenter** — Scales a geometry in 3D space around a specified center point.

#### Synopsis

geometry **CG\_3DScaleAroundCenter**(geometry geom, float8 factorX, float8 factorY, float8 factorZ, float8 centerX, float8 centerY, float8 centerZ);

SQL

Scales the input geometry *geom* by different factors along the X, Y, and Z axes, relative to a specified center point (*centerX*, *centerY*, *centerZ*).

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.



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

SQL

```
SELECT ST_AsText(CG_3DScaleAroundCenter('POINT(2 2 2)', 0.5, 0.5, 0.5, 1, 1, 1));
-- Result: POINT(1.5 1.5 1.5)
```

SQL

[CG\\_Scale](#), [CG\\_3DScale](#)

### 8.3.50 CG\_Translate

**CG\_Translate** — Translates (moves) a geometry by given offsets in 2D space.

#### Synopsis

geometry **CG\_Translate**(geometry geom, float8 deltaX, float8 deltaY);

☒☒

Translates the input geometry *geom* by adding *deltaX* to the X coordinates and *deltaY* to the Y coordinates. Z coordinates are dropped.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.

☒☒

```
SELECT ST_AsText(CG_Translate('LINESTRING(1 1, 2 2)', 1, -1));
-- Result: LINESTRING(2 0, 3 1)
```

☒☒

**CG\_3DTranslate**, **ST\_Translate**

### 8.3.51 CG\_3DTranslate

**CG\_3DTranslate** — Translates (moves) a geometry by given offsets in 3D space.

#### Synopsis

geometry **CG\_3DTranslate**(geometry geom, float8 deltaX, float8 deltaY, float8 deltaZ);

☒☒

Translates the input geometry *geom* by adding *deltaX* to the X coordinates, *deltaY* to the Y coordinates, and *deltaZ* to the Z coordinates.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.



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

☒☒

```
SELECT ST_AsText(CG_3DTranslate('POINT(1 1 1)', 1, -1, 2));
-- Result: POINT(2 0 3)
```

☒☒

**CG\_Translate**, **ST\_Translate**

### 8.3.52 CG\_Simplify

**CG\_Simplify** — Reduces the complexity of a geometry while preserving essential features and Z/M values.



## Synopsis

geometry **CG\_Simplify**(geometry geom, double precision threshold, boolean preserveTopology = false);

**⚠**

Simplifies a geometry using SFCGAL's simplification algorithm, which reduces the number of points or vertices while preserving the essential features of the geometry. This function preserves Z and M values during simplification.

The algorithm is based on constrained triangulation and uses the [CGAL Polyline Simplification 2](#) library with additional handling to preserve Z and M coordinates. When topology is preserved and geometries intersect, Z and M values are interpolated at intersection points.

This function works well with 3D terrain-like geometries (2.5D) but is not designed for vertical surfaces like walls.

Availability: 3.6.0 - requires SFCGAL >= 2.1.0



This method needs SFCGAL backend.



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



This function supports M coordinates.

## Parameters

**geom** Input geometry

**threshold** Maximum distance threshold (in geometry unit) for simplification. The higher this value, the more simplified the resulting geometry will be.

**preserveTopology** If set to true, the function ensures that the topology of the geometry is preserved. When geometries intersect in this mode, Z and M values at intersection points are interpolated. The default value is false.

## Return Value

Returns a simplified geometry with preserved Z and M values.

**⚠**

```
-- Simplify a polygon with a threshold of 0.5
SELECT ST_AsText(CG_Simplify(ST_GeomFromText('POLYGON((0 0, 0 1, 0.1 1, 0.2 1, 0.3 1, 0.4 1, 0.5 1, 1 1, 1 0, 0 0))'), 0.5));

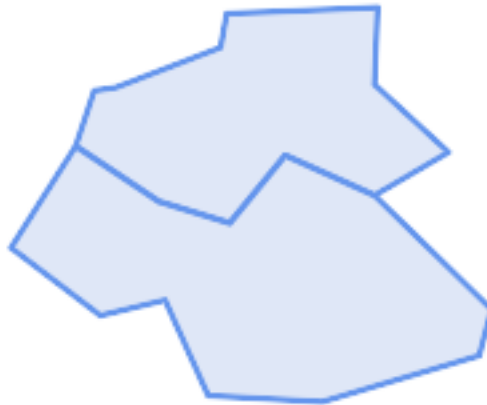
-- Simplify a 3D terrain geometry while preserving topology and Z values
SELECT ST_AsText(CG_Simplify(ST_GeomFromText('LINESTRING Z(0 0 0, 0 1 1, 0.1 1 1, 0.2 1 1, 0.3 1 1, 1 1 2)'), 0.2, true));

-- Simplify a geometry with both Z and M values
SELECT ST_AsText(CG_Simplify(ST_GeomFromText('LINESTRING ZM(0 0 0 1, 0 1 1 2, 0.1 1 1 3, 0.2 1 1 4, 0.3 1 1 5, 1 1 2 6)'), 0.2));
```

```
-- Simplify two geometry together preserving Z and M values, without topology
SELECT ST_AsText(CG_Simplify('GEOMETRYCOLLECTION ZM(LINESTRING ZM(-1 -1 3 4, 0 0 10 100, 1
1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150), POLYGON ZM((0 0 10 100, 1
1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150, 0 0 10 100)))', 2, false));

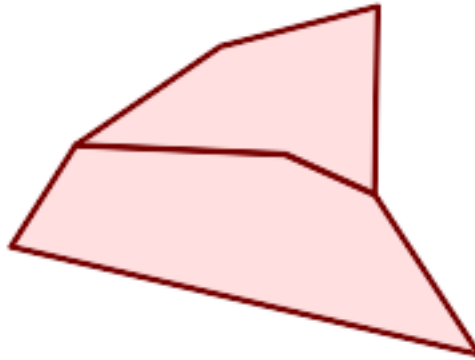
-- Simplify two geometry together preserving Z and M values, with topology
SELECT ST_AsText(CG_Simplify('GEOMETRYCOLLECTION ZM(LINESTRING ZM(-1 -1 3 4, 0 0 10 100, 1
1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150), POLYGON ZM((0 0 10 100, 1
1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150, 0 0 10 100)))', 2, true));

WITH depts_pds as (SELECT ST_GeomFromText('GEOMETRYCOLLECTION(
POLYGON((88.46 158.85,90.77 171.54,147.31 173.85,146.15 145,173.85 119.62,146.15
103.46,112.69 118.46,91.92 93.08,65.38 101.15,34.23 121.92,41.15 142.69,49.23
143.85,88.46 158.85)),
POLYGON((112.69 118.46,146.15 103.46,190 60.77,185.38 43.46,126.54 26.15,83.85 28.46,67.69
64.23,43.46 58.46,10 83.85,34.23 121.92,65.38 101.15,91.92 93.08,112.69 118.46)))
') as geom)
SELECT geom FROM depts_pds;
```



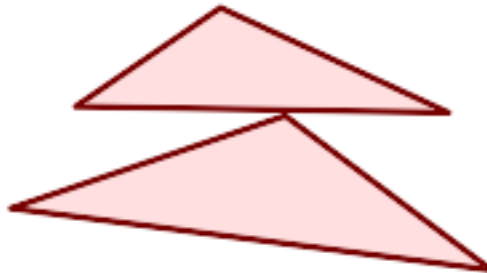
*Originals geometries*

```
WITH depts_pds as (SELECT ST_GeomFromText('GEOMETRYCOLLECTION(
POLYGON((88.46 158.85,90.77 171.54,147.31 173.85,146.15 145,173.85 119.62,146.15
103.46,112.69 118.46,91.92 93.08,65.38 101.15,34.23 121.92,41.15 142.69,49.23
143.85,88.46 158.85)),
POLYGON((112.69 118.46,146.15 103.46,190 60.77,185.38 43.46,126.54 26.15,83.85 28.46,67.69
64.23,43.46 58.46,10 83.85,34.23 121.92,65.38 101.15,91.92 93.08,112.69 118.46)))
') as geom)
SELECT (ST_Dump(CG_Simplify(geom, 0.5, true))).geom FROM depts_pds;
```



*Simplification with 0.5 and topology preserved*

```
WITH depts_pds as (SELECT ST_GeomFromText('GEOMETRYCOLLECTION(
POLYGON((88.46 158.85,90.77 171.54,147.31 173.85,146.15 145,173.85 119.62,146.15  ←
103.46,112.69 118.46,91.92 93.08,65.38 101.15,34.23 121.92,41.15 142.69,49.23  ←
143.85,88.46 158.85)),
POLYGON((112.69 118.46,146.15 103.46,190 60.77,185.38 43.46,126.54 26.15,83.85 28.46,67.69  ←
64.23,43.46 58.46,10 83.85,34.23 121.92,65.38 101.15,91.92 93.08,112.69 118.46)))
') as geom)
SELECT (ST_Dump(CG_Simplify(geom, 0.5, false))).geom FROM depts_pds;
```



*Simplification with 0.5 without topology preservation*



[ST\\_Simplify](#), [ST\\_SimplifyPreserveTopology](#)

### 8.3.53 CG\_3DAlphaWrapping

**CG\_3DAlphaWrapping** — Computes a 3D Alpha-wrapping strictly enclosing a geometry.

## Synopsis

geometry **CG\_3DAlphaWrapping**(geometry geom, integer relative\_alpha, integer relative\_offset);

☒☒

Computes the **3D Alpha Wrapping** of the points in a geometry. An alpha wrapping is a watertight and orientable surface mesh that strictly encloses the input. It can be seen as an extension or refinement of an **alpha-shape**.

The `relative_alpha` parameter controls which features will appear in the output. It can have values from 0 to infinity. Smaller `relative_alpha` values result in simpler outputs, but they are less accurate representations of the original input.

The `relative_offset` parameter controls the tightness of the result. It can have values from 0 to infinity. If this parameter is set to 0, its value is automatically determined based on the `relative_alpha` parameter.

Availability: 3.6.0 - requires SFCGAL >= 2.1.0



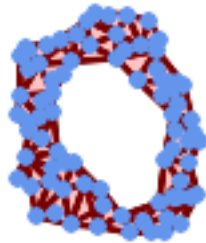
This method needs SFCGAL backend.



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

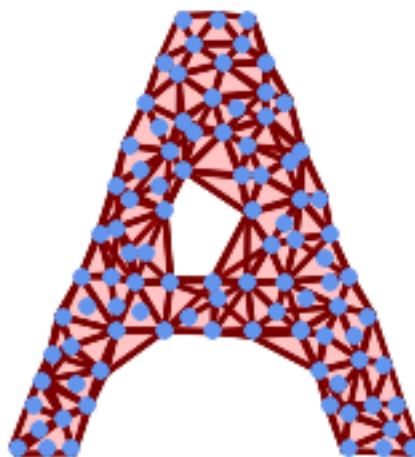
☒☒

```
SELECT CG_3DAlphaWrapping('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,10);
```

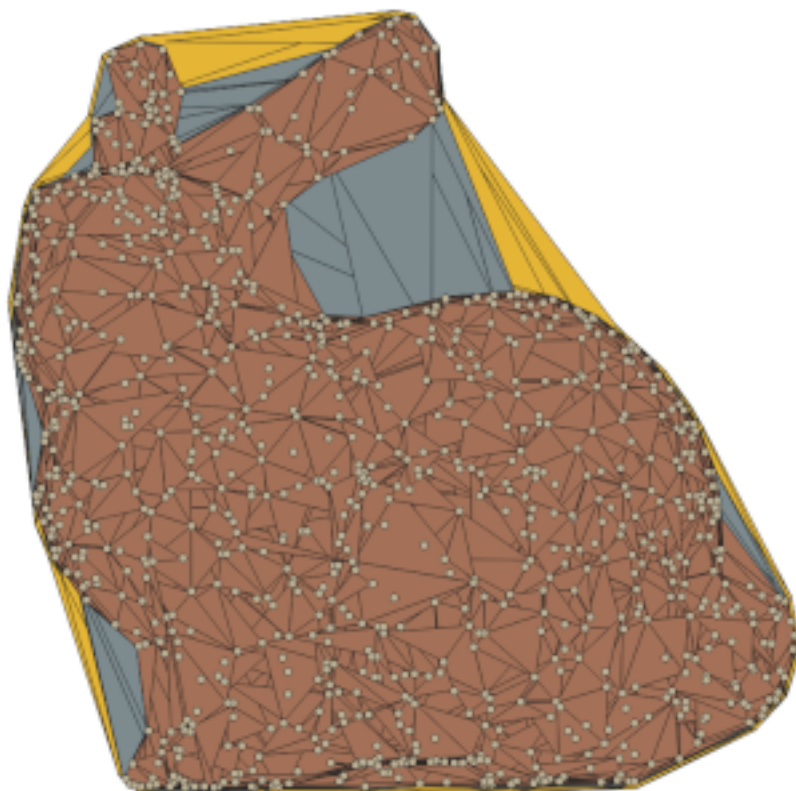


*Alpha wrapping of a MultiPoint (same example As [CG\\_OptimalAlphaShape](#))*

```
SELECT CG_3DAlphaWrapping('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,14);
```



*Alpha wrapping of a MultiPoint (same example as [ST\\_ConcaveHull](#))*



*Effect of the `relative_alpha` parameter with values 5, 10 and 20. A value of 5 results in a coarse output. Increasing the parameter up to 20 significantly improves the precision and granularity of the result.*

☒☒

CG\_AlphaShape

# Chapter 9

 (topology)

[illegible]

Sandro Santilli's presentation at PostGIS Day Paris 2011 conference gives a good synopsis of PostGIS Topology and where it is headed [Topology with PostGIS 2.0 slide deck](#).

Vincent Picavet provides a good synopsis and overview of what is Topology, how is it used, and various FOSS4G tools that support it in [PostGIS Topology PGConf EU 2012](#).

GIS US Census Topologically Integrated Geographic Encoding and Referencing System (TIGER) PostGIS Topology Load Tiger.

PostGIS 2.0.0 支持 PostGIS 2.0.0 版本, 支持 PostGIS 2.0.0 版本. PostGIS 2.0.0 支持, 支持 PostGIS 2.0.0 版本, 支持 PostGIS 2.0.0 版本. SQL-MM 支持 PostGIS 2.0.0 版本.

PostGIS Topology Wiki

□□□□□□□□□□□□□□ topology □□□□□□□□□□□□□□.

**SQL/MM** **ST\_**, **PostGIS**

Topology support is build by default starting with PostGIS 2.0, and can be disabled specifying `--without-topology` configure option at build time as described in [Chapter 2](#)

**9.1** ☐ ☐ ☐ ☐

### 9.1.1 getfaceedges\_returntype

`getfaceedges` `returntype` — A composite type that consists of a sequence number and an edge number.



A composite type that consists of a sequence number and an edge number. This is the return type for `ST_GetFaceEdges` and `GetNodeEdges` functions.

1. sequence `[SRID]`: `[SRID]` SRID `[topology.topology]` `[topology.topology]`.
2. edge `[edge_id]`: `[edge_id]`.

### 9.1.2 TopoGeometry

TopoGeometry — A composite type representing a topologically defined geometry.



0000000000, 00000000 ID 00000000000000000000. TopoGeometry 0000  
 0000 topology\_id, layer\_id, id, type 00000000.

1. **topology\_id** integer: integer SRID integer topology.topology integer  
integer.
2. **layer\_id** integer: TopoGeometry integer layer\_id integer. topology\_id integer  
layer\_id integer topology.layers integer (unique reference) integer.
3. **id** integer: integer, integer.
4. 1 integer 4 integer type integer. 1: [integer] integer, 2: [integer] integer, 3: [integer] integer, 4:  
integer.

☐ ☐ ☐ ☐ ☐

[illegible]

|                                                                                       |                                                  |
|---------------------------------------------------------------------------------------|--------------------------------------------------|
| <div> <div> <div></div> <div></div> <div></div> <div></div> <div></div> </div> </div> | <div> <div></div> <div></div> </div>             |
| <div> <div></div> <div></div> </div>                                                  | <div> <div></div> <div></div> <div></div> </div> |



## CreateTopoGeom

### 9.1.3 validate\_topology returntype

`validateTopology_returntype` — A composite type that consists of an error message and `id1` and `id2` to denote location of error. This is the return type for `ValidateTopology`.



2 `ValidateTopology` ID `id1` `id2`

- error** `varchar(255)`: `coincident nodes(id1), edge crosses node(id1), edge not simple(id1), edge end node geometry mismatch(id1), edge start node geometry mismatch(id1), face overlaps face(id1), face within face(id1)`
- id1** `varchar(255)`: `edge`/`face`/`node` `id1`
- id2** `varchar(255)`: `id2` `id1`



图元

[ValidateTopology](#)

## 9.2 图元

### 9.2.1 TopoElement

TopoElement — 图元是 TopoGeometry 对象的基本组成单元，由 2 个属性组成。

图元

图元是 [TopoGeometry](#) 对象的基本组成单元，由 2 个属性组成。

图元 TopoGeometry 对象，图元是图元对象 (topological primitive) 对象的基本组成单元 (1: node, 2: edge, 3: face) 对象。图元 TopoGeometry 对象是图元对象的基本组成单元。图元 TopoGeometry 对象是图元对象的基本组成单元。



#### Note

图元 TopoGeometry 对象，图元 TopoGeometry 对象，图元 TopoGeometry 对象是图元对象的基本组成单元。图元 TopoGeometry 对象是图元对象的基本组成单元。

图元

```
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: topoelement 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"
```

图元

[GetTopoGeomElements](#), [TopoElementArray](#), [TopoGeometry](#), [TopoGeom\\_addElement](#), [TopoGeom\\_remElement](#)

### 9.2.2 TopoElementArray

TopoElementArray — An array of TopoElement objects.

国国

1 国国国 TopoElement 国国国国国, 国国国 TopoElement 国国国国国国国国国国国国国国.

国国

```
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"
```

国国

[TopoElement](#), [GetTopoGeomElementArray](#), [TopoElementArray\\_Agg](#)

## 9.3 国国国 TopoGeometry 国国

### 9.3.1 AddTopoGeometryColumn

AddTopoGeometryColumn — 国国国国国 TopoGeometry 国国国国国, topology.layer 国国国国国国国国国国国, 国 layer\_id 国国国国国.

#### Synopsis

```
integer AddTopoGeometryColumn(name topology_name, name schema_name, name table_name,
name column_name, varchar feature_type, integer child_layer);
integer AddTopoGeometryColumn(name topology_name, regclass tab, name column_name, integer
layer_id, varchar feature_type, integer child_layer);
```

国国

国 TopoGeometry 国国国国国国国国国国国国国国. TopoGeometry 国国国国国国国国国国国国国国国国国国国国. 国国国国国国国国国国国国国国国国国国国国. AddTopoGeometryColumn() 国国国国国国国国国国国国国国国国国国国国:

topology.layer 函数。

[child\_layer] (NULL) 值, 函数 (函数) 返回 TopoGeometry 值。函数, 函数 (child\_layer 返回 TopoGeometry 值) 返回 TopoGeometry 值。

函数 (AddTopoGeometryColumn 函数 ID 函数) 返回 TopoGeometry 值。

Valid feature\_types are: POINT, MULTIPOINT, LINE, MULTILINE, POLYGON, MULTIPOLYGON, COLLECTION

Availability: 1.1

例

```
-- 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');
```

例

[DropTopoGeometryColumn](#), [toTopoGeom](#), [CreateTopology](#), [CreateTopoGeom](#)

### 9.3.2 RenameTopoGeometryColumn

RenameTopoGeometryColumn — Renames a topogeometry column

#### Synopsis

topology.layer **RenameTopoGeometryColumn**(regclass layer\_table, name feature\_column, name new\_name)

例

This function changes the name of an existing TopoGeometry column ensuring metadata information about it is updated accordingly.

Availability: 3.4.0

例

```
SELECT topology.RenameTopoGeometryColumn('public.parcels', 'topogeom', 'tgeom');
```

文档

[AddTopoGeometryColumn, RenameTopology](#)

### 9.3.3 DropTopology

**DropTopology** — 删除 topology.topology 表中的 topology 记录，并删除 geometry\_columns 表中的相应记录。

#### Synopsis

```
integer DropTopology(varchar topology_schema_name);
```

文档

**DropTopology** 删除 topology.topology 表中的 topology 记录，并删除 geometry\_columns 表中的相应记录。该函数在 topology 表中删除记录，并在 geometry\_columns 表中删除相应的记录。该函数在 topology 表中删除记录，并在 geometry\_columns 表中删除相应的记录。

Availability: 1.1

文档

ma\_topo 删除 topology.topology 表中的 topology 记录，并删除 geometry\_columns 表中的相应记录。

```
SELECT topology.DropTopology('ma_topo');
```

文档

[DropTopoGeometryColumn](#)

### 9.3.4 RenameTopology

**RenameTopology** — 重命名 topology

#### Synopsis

```
varchar RenameTopology(varchar old_name, varchar new_name);
```

文档

重命名 topology 模式，更新其在 topology.topology 表中的元数据记录。

Availability: 3.4.0

例

Rename a topology from topo\_stage to topo\_prod.

```
SELECT topology.RenameTopology('topo_stage', 'topo_prod');
```

例

[CopyTopology](#), [RenameTopoGeometryColumn](#)

### 9.3.5 DropTopoGeometryColumn

**DropTopoGeometryColumn** — schema\_name テーブル名 table\_name Topogeometry 列 topology.layer から削除する。

#### Synopsis

text **DropTopoGeometryColumn**(varchar schema\_name, varchar table\_name, varchar column\_name);

例

schema\_name テーブル名 Topogeometry 列 topology.layer から削除する。例: schema\_name テーブル名 Topogeometry 列 topology.layer から削除する。 NULL 列を削除する。

Availability: 1.1

例

```
SELECT topology.DropTopoGeometryColumn('ma_topo', 'parcel_topo', 'topo');
```

例

[AddTopoGeometryColumn](#)

### 9.3.6 Populate\_Topology\_Layer

**Populate\_Topology\_Layer** — Adds missing entries to topology.layer table by reading metadata from topo tables.

#### Synopsis

setof record **Populate\_Topology\_Layer**();

¶¶

Adds missing entries to the `topology.layer` table by inspecting topology constraints on tables. This function is useful for fixing up entries in topology catalog after restores of schemas with topo data.

It returns the list of entries created. Returned columns are `schema_name`, `table_name`, `feature_column`.

2.3.0 ¶¶¶¶¶¶¶¶¶¶.

¶¶

```
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)
```

¶¶

**AddTopoGeometryColumn**

### 9.3.7 TopologySummary

**TopologySummary** — Takes a topology name and provides summary totals of types of objects in topology.

#### Synopsis

text **TopologySummary**(varchar topology\_schema\_name);

Takes a topology name and provides summary totals of types of objects in topology.

2.0.0

```
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
```

Topology\_Load\_Tiger

9.3.8 ValidateTopology

ValidateTopology — Returns a set of validate\_topology\_returntype objects detailing issues with topology.

Synopsis

setof validate\_topology\_returntype **ValidateTopology**(varchar toponame, geometry bbox);

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:

|                         | id1                       | id2                        | Meaning                                                    |
|-------------------------|---------------------------|----------------------------|------------------------------------------------------------|
| coincident nodes        | Identifier of first node. | Identifier of second node. | Two nodes have the same geometry.                          |
| edge crosses node(<br>) | Identifier of the edge.   | Identifier of the node.    | An edge has a node in its interior. See <b>ST_Relate</b> . |

| 国国                                                | id1                                                                                    | id2                                     | Meaning                                                                                                                                                      |
|---------------------------------------------------|----------------------------------------------------------------------------------------|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| invalid edge(国国国国国国国国国国)                          | Identifier of the edge.                                                                |                                         | An edge geometry is invalid. See <a href="#">ST_IsValid</a> .                                                                                                |
| edge not simple(国国国国国国国国国国)                       | Identifier of the edge.                                                                |                                         | An edge geometry has self-intersections. See <a href="#">ST_IsSimple</a> .                                                                                   |
| edge crosses edge(国国国国国国国国国国国国)                   | Identifier of first edge.                                                              | Identifier of second edge.              | Two edges have an interior intersection. See <a href="#">ST_Relate</a> .                                                                                     |
| edge start node geometry mismatch(国国国国国国国国国国国国国国) | 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> . |
| edge end node geometry mismatch(国国国国国国国国国国国国国国)   | 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> .      |
| face without edges(国国国国国国国国国国)                    | Identifier of the orphaned face.                                                       |                                         | No edge reports an existing face on either of its sides (left_face, right_face).                                                                             |
| face has no rings(国国国国国国国国国国)                     | 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.                                                                             |
| 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.                                           |



|                                         | id1                                                                                    | id2                                                                                    | Meaning                                                                                                                                                                                 |
|-----------------------------------------|----------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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.                                                                                                                   |

1.0.0 [Topology\\_ValidateTopology](#).

[Topology\\_ValidateTopology](#): 2.0.0 [Topology\\_ValidateTopology](#), [Topology\\_ValidateTopology](#) (false positive) [Topology\\_ValidateTopology](#).

[Topology\\_ValidateTopology](#): 2.2.0 [Topology\\_ValidateTopology](#)'edge crosses node' [Topology\\_ValidateTopology](#) id1 id2 [Topology\\_ValidateTopology](#).

Changed: 3.2.0 added optional bbox parameter, perform face labeling and edge linking checks.

[Topology\\_ValidateTopology](#)

```
SELECT * FROM topology.ValidateTopology('ma_topo');
      error      | id1 | id2
-----+-----+-----
face without edges |    1 |
```

[Topology\\_ValidateTopology](#)

[Topology\\_ValidateTopology](#), [Topology\\_LoadTiger](#)

### 9.3.9 ValidateTopologyRelation

ValidateTopologyRelation — Returns info about invalid topology relation records

#### Synopsis

setof record **ValidateTopologyRelation**(varchar toponame);

☒☒

Returns a set records giving information about invalidities in the relation table of the topology.

Availability: 3.2.0

☒☒

**ValidateTopology**

### 9.3.10 ValidateTopologyPrecision

ValidateTopologyPrecision — Returns non-precise vertices in the topology.

#### Synopsis

geometry **ValidateTopologyPrecision**(name toponame, geometry bbox, float8 gridSize);

☒☒

Returns all vertices that are not rounded to the topology or given `gridSize` as a puntal geometry, optionally limiting the check to the area specified by the `bbox` parameter.

Availability: 3.6.0

☒☒

```
SELECT ST_AsEWKT(g) FROM
  topology.ValidateTopologyPrecision(
    'city_data',
    gridSize =
> 2,
    bbox =
> ST_MakeEnvelope(0,0,20,20)
  ) g;
      st_asewkt
-----
MULTIPOINT(9 6,9 14)
(1 row)
```

☒☒

**MakeTopologyPrecise**

---

### 9.3.11 MakeTopologyPrecise

MakeTopologyPrecise — Snap topology vertices to precision grid.

#### Synopsis

```
void MakeTopologyPrecise(name toponame, geometry bbox, float8 gridSize);
```

⊠

Snaps all vertices of a topology to the topology precision grid or to the grid whose size is specified with the `gridSize` parameter, optionally limiting the operation to the objects intersecting the area specified by the `bbox` parameter.



#### Note

Snapping could make the topology invalid, so it is recommended to check the outcome of operation with [ValidateTopology](#).

Availability: 3.6.0

⊠

```
SELECT topology.MakeTopologyPrecise(
    'city_data',
    gridSize =
> 2
);
maketopologyprecise
-----
(1 row)
```

⊠

[ValidateTopologyPrecision](#), [ValidateTopology](#)

### 9.3.12 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);
```

☒☒

Takes a topology identifier or the identifier of a topology-related object and returns a topology.topology record.

Availability: 3.2.0

☒☒

```
SELECT name(findTopology('features.land_parcels', 'feature'));
       name
-----
city_data
(1 row)
```

☒☒

**FindLayer**

### 9.3.13 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);
```

☒☒

Takes a layer identifier or the identifier of a topology-related object and returns a topology.layer record.

Availability: 3.2.0

☒☒

```
SELECT layer_id(findLayer('features.land_parcels', 'feature'));
       layer_id
-----
            1
(1 row)
```

☒☒

**FindTopology**

### 9.3.14 TotalTopologySize

**TotalTopologySize** — Total disk space used by the specified topology, including all indexes and TOAST data.

#### Synopsis

```
int8 TotalTopologySize(name toponame);
```

⊠

Takes a topology name and provides the total disk space used by all its tables, including indexes and TOAST data.

Availability: 3.6.0

⊠

```
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
```

⊠

**Topology\_Load\_Tiger**

### 9.3.15 UpgradeTopology

**UpgradeTopology** — Upgrades the specified topology to support large ids (int8) for topology and primitive ids.

#### Synopsis

```
void UpgradeTopology(name toponame);
```

☒☒

Takes a topology name and upgrades it to support large ids (int8) for topology and primitive ids. The function upgrades the following: - face (face\_id column from int4 to int8, face\_id\_seq from int4 to int8) - node (node\_id column from int4 to int8, containing\_face column from int4 to int8, node\_id\_seq from int4 to int8) - edge\_data (edge\_id column from int4 to int8, edge\_data\_edge\_id\_seq from int4 to int8, left\_face and right\_face columns from int4 to int8, start\_node and end\_node columns from int4 to int8, next\_left\_edge and next\_right\_edge columns from int4 to int8) - relation (topogeo\_id column from int4 to int8, element\_id from int4 to int8) - topology (useslargeids column set to true)

Availability: 3.6.0

☒☒

```
SELECT topology.upgradetopology('city_data');
```

## 9.4 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.

## 9.5 ☒☒☒☒☒

### 9.5.1 CreateTopology

CreateTopology — Creates a new topology schema and registers it in the topology.topology table.

#### Synopsis

integer **CreateTopology**(name topology\_schema\_name, integer srid, double precision prec, boolean hasz, integer topoid, boolean useslargeids);

☒☒

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 SRID defaults to -1 (unknown) if not specified.

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.

`topoid` optional explicit identifier (allows deterministic topology id assignment, needs to be unique)

`useslargeids` optional, defaults to false. If true, the topology will be created to support large ids (int8) for topology and primitive ids.

This is similar to the SQL/MM **ST\_InitTopoGeo** but has more functionality.

Availability: 1.1

Enhanced: 2.0 added the signature accepting `hasZ`

☒☒

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
```

☒☒

Section [4.5](#), **ST\_InitTopoGeo**, **Topology\_Load\_Tiger**

## 9.5.2 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);
```

☒☒

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.

2.0.0

Make a backup of a topology called `ma_topo`.

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_backup');
```

Section [4.5, CreateTopology, RenameTopology](#)

### 9.5.3 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);

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 and Topo-Net 3: Routine Details: X.3.17

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation
           astopocreation
```

```
-----
Topology-Geometry 'topo_schema_to_create' (id:7) created.
```

[CreateTopology](#)



## 9.5.4 ST\_CreateTopoGeo

ST\_CreateTopoGeo — Creates a topology from a geometry collection.

### Synopsis

text **ST\_CreateTopoGeo**(varchar atopoology, geometry acollection);

2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details -- X.3.18

```
-- 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');
```

[TopoGeo\\_LoadGeometry](#), [AddTopoGeometryColumn](#), [CreateTopology](#), [DropTopology](#)

### 9.5.5 TopoGeo\_AddPoint

TopoGeo\_AddPoint — `bigint TopoGeo_AddPoint(varchar atopolgy, geometry apoint, float8 tolerance)` (split) `bigint`.

#### Synopsis

`bigint TopoGeo_AddPoint(vchar atopolgy, geometry apoint, float8 tolerance);`

`TopoGeo_AddPoint`

Adds a point to an existing topology and returns its identifier. The given point will snap to existing nodes or edges within given tolerance. An existing edge may be split by the snapped point.

2.0.0 `TopoGeo_AddPoint`.

`TopoGeo_AddPoint`

[TopoGeo\\_AddLineString](#), [TopoGeo\\_AddPolygon](#), [TopoGeo\\_LoadGeometry](#), [AddNode](#), [CreateTopology](#)

### 9.5.6 TopoGeo\_AddLineString

TopoGeo\_AddLineString — Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.

#### Synopsis

`SETOF bigint TopoGeo_AddLineString(vchar atopolgy, geometry aline, float8 tolerance);`

`TopoGeo_AddLineString`

Adds a linestring to an existing topology and returns a set of signed edge identifiers forming it up (negative identifies mean the edge goes in the opposite direction of the input linestring). The given line will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the line. New nodes and faces may be added.



#### Note

Updating statistics about topologies being loaded via this function is up to caller, see [maintaining statistics during topology editing and population](#).

2.0.0 `TopoGeo_AddLineString`.

Enhanced: 3.2.0 added support for returning signed identifier.

`TopoGeo_AddLineString`

[TopoGeo\\_AddPoint](#), [TopoGeo\\_AddPolygon](#), [TopoGeo\\_LoadGeometry](#), [AddEdge](#), [CreateTopology](#)

## 9.5.7 TopoGeo\_AddPolygon

**TopoGeo\_AddPolygon** — Adds a polygon to an existing topology using a tolerance and possibly splitting existing edges/faces. Returns face identifiers.

### Synopsis

SETOF bigint **TopoGeo\_AddPolygon**(varchar atopology, geometry apoly, float8 tolerance);



Adds a polygon to an existing topology and returns a set of face identifiers forming it up. The boundary of the given polygon will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the boundary of the new polygon.



#### Note

Updating statistics about topologies being loaded via this function is up to caller, see [maintaining statistics during topology editing and population](#).

2.0.0 .



[TopoGeo\\_AddPoint](#), [TopoGeo\\_AddLineString](#), [TopoGeo\\_LoadGeometry](#), [AddFace](#), [CreateTopology](#)

## 9.5.8 TopoGeo\_LoadGeometry

**TopoGeo\_LoadGeometry** — Load a geometry into an existing topology, snapping and splitting as needed.

### Synopsis

void **TopoGeo\_LoadGeometry**(varchar atopology, geometry ageom, float8 tolerance);



Loads a geometry into an existing topology. The given geometry will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split as a consequence of the load.



#### Note

Updating statistics about topologies being loaded via this function is up to caller, see [maintaining statistics during topology editing and population](#).

Availability: 3.5.0

関数

[TopoGeo\\_AddPoint](#), [TopoGeo\\_AddLineString](#), [TopoGeo\\_AddPolygon](#), [CreateTopology](#)

## 9.6 関数一覧

### 9.6.1 ST\_AddIsoNode

`ST_AddIsoNode` — 孤立 (isolated) ノード ID を追加。 NULL 面、面、線、多角形。

#### Synopsis

`bigint ST_AddIsoNode(varchar atopology, bigint aface, geometry apoint);`

関数

`atopology` 面 `aface ID(faceid)` ノード `apoint` ノード ID(`nodeid`) を追加。

`SRID` (SRID) を指定、`apoint` を指定、 NULL 面、面、線、多角形 (面、線、多角形) を指定。 孤立ノードを追加。

`aface` NULL 面 `apoint` ノード、面、線、多角形。

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X+1.3.1

関数

関数

[AddNode](#), [CreateTopology](#), [DropTopology](#), [ST\\_Intersects](#)

### 9.6.2 ST\_AddIsoEdge

`ST_AddIsoEdge` — ノード `anode` と `anothernode` の `alinestring` を追加。 ID を指定。

#### Synopsis

`bigint ST_AddIsoEdge(varchar atopology, bigint anode, bigint anothernode, geometry alinestring);`

¶¶

¶¶¶¶¶¶¶¶¶¶ anode ¶ anothernode ¶¶¶¶ alinestring ¶¶¶¶¶¶¶¶¶¶¶¶  
¶¶¶¶¶¶¶¶ ID(edgeid) ¶¶¶¶¶.

alinestring ¶¶¶¶¶¶¶¶¶¶¶ (SRID) ¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶ NULL ¶¶, ¶¶¶  
¶¶¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶.

alinestring ¶ anode ¶ anothernode ¶¶¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶.

anode ¶ anothernode ¶ alinestring ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

Availability: 1.1

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4

¶¶

¶¶

[ST\\_AddIsoNode](#), [ST\\_IsSimple](#), [ST\\_Within](#)

### 9.6.3 ST\_AddEdgeNewFaces

ST\_AddEdgeNewFaces — ¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶¶  
¶¶ 2 ¶¶¶¶¶¶¶.

#### Synopsis

bigint **ST\_AddEdgeNewFaces**(varchar atopology, bigint anode, bigint anothernode, geometry acurve);

¶¶

¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶ 2 ¶¶¶¶¶¶¶. ¶¶¶  
¶¶¶¶¶ ID ¶¶¶¶¶.

¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

¶¶¶¶ NULL ¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶¶¶ (¶¶¶¶¶¶¶¶¶¶ node ¶¶¶¶¶¶¶¶  
¶¶), acurve ¶ LINESTRING ¶¶¶¶, anode ¶ anothernode ¶ acurve ¶¶¶¶¶¶¶¶¶¶¶¶  
¶¶¶¶¶¶.

acurve ¶¶¶¶¶¶¶¶¶¶¶ (SRID) ¶¶¶¶¶¶¶¶¶¶¶.

2.0 ¶¶¶¶¶¶¶¶¶¶.

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.12

¶¶

¶¶

[ST\\_RemEdgeNewFace](#)

[ST\\_AddEdgeModFace](#)

## 9.6.4 ST\_AddEdgeModFace

`ST_AddEdgeModFace` — 国国国国国国国国, 国国国国国国国国国国国国国国, 国国国国国国国国国国国国国国国国.

### Synopsis

`bigint ST_AddEdgeModFace(varchar atopology, bigint anode, bigint anothernode, geometry acurve);`

国国

国国国国国国国国, 国国国国国国国国国国国国国国国国, 国国国国国国国国国国国国国国国国.



#### Note

国国国国, 国国国国国国国国国国国国国国国国. 国国国国国 (国国国) 国国国 (universe face) 国国国国国国国国国国国国国国.

国国国国国国国国 ID 国国国国国.

国国国国国国国国国国国国国国国国国国国国国国国国国国.

国国国国 NULL 国国, 国国国国国国国国国国国国 (国国国国国国国国国国 node 国国国国国国国国国国), `acurve` 国 `LINESTRING` 国国国国, `anode` 国 `anothernode` 国 `acurve` 国国国国国国国国国国国国国国国国.

`acurve` 国国国国国国国国国国国国 (SRID) 国国国国国国国国国国国国.

2.0 国国国国国国国国国国国国.



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13

国国

国国

[ST\\_RemEdgeModFace](#)

[ST\\_AddEdgeNewFaces](#)

## 9.6.5 ST\_RemEdgeNewFace

`ST_RemEdgeNewFace` — 国国国国国国国国, 国国国国国国国国国国国国国国国国, 国国国国国国国国国国国国国国国国.

### Synopsis

`bigint ST_RemEdgeNewFace(varchar atopology, bigint anedge);`

国国

国国国国国国, 国国国国国国国国国国国国国国, 国国国国国国国国国国国国国国国国.

国国国国国国 ID 国国国国, 国国国国国国国国国国国 NULL 国国国国. 国国国国国国国国国国国国国, 国国国国, 国 (国国国国国国国国国国国国国国国国国国) 国国国国国国国国国国国国国.

国国国国国国国国国国国国国国国国国国国国国国.

Refuses to remove an edge participating in the definition of an existing TopoGeometry. Refuses to heal two faces if any TopoGeometry is defined by only one of them (and not the other).

国国国国 NULL 国国, 国国国国国国国国国国国 (国国国国国国国国国国 edge 国国国国国国国国), 国国国国国国国国国国国国国国国国国国.

2.0 国国国国国国国国国国.



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14

国国

国国

**ST\_RemEdgeModFace**

**ST\_AddEdgeNewFaces**

## 9.6.6 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

bigint **ST\_RemEdgeModFace**(varchar atopology, bigint anedge);

国国

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.

国国国国国国国国国国国国国国国国国国国国国国.

Refuses to remove an edge participating in the definition of an existing TopoGeometry. Refuses to heal two faces if any TopoGeometry is defined by only one of them (and not the other).

国国国国 NULL 国国, 国国国国国国国国国国国 (国国国国国国国国国国 edge 国国国国国国国国), 国国国国国国国国国国国国国国国国国国.

2.0 国国国国国国国国国国.



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.15





## 9.6.8 ST\_ModEdgeSplit

**ST\_ModEdgeSplit** — Splits an edge in a topology, returning the id of the new edge and the id of the deleted node.

### Synopsis

**bigint ST\_ModEdgeSplit**(varchar atopology, bigint anedge, geometry apoint);

**ST**

**ST** — Splits an edge in a topology, returning the id of the new edge and the id of the deleted node. **ST** — Splits an edge in a topology, returning the id of the new edge and the id of the deleted node.

Availability: 1.1

**ST**: 2.0 **ST**, **ST** **ST\_ModEdgesSplit** **ST**.



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

**ST**

```
-- 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
```

**ST**

**ST\_NewEdgesSplit**, **ST\_ModEdgeHeal**, **ST\_NewEdgeHeal**, **AddEdge**

## 9.6.9 ST\_ModEdgeHeal

**ST\_ModEdgeHeal** — Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node.

### Synopsis

**bigint ST\_ModEdgeHeal**(varchar atopology, bigint anedge, bigint anotheredge);

⊠

Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node. Updates all existing joined edges and relationships accordingly.

2.0 ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠.

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

⊠

[ST\\_ModEdgeSplit](#) [ST\\_NewEdgesSplit](#)

### 9.6.10 ST\_NewEdgeHeal

**ST\_NewEdgeHeal** — Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided.

#### Synopsis

**bigint ST\_NewEdgeHeal**(varchar atopology, bigint anedge, bigint anotheredge);

⊠

Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided. Returns the id of the new edge replacing the healed ones. Updates all existing joined edges and relationships accordingly.

2.0 ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠.

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

⊠

[ST\\_ModEdgeHeal](#) [ST\\_ModEdgeSplit](#) [ST\\_NewEdgesSplit](#)

### 9.6.11 ST\_MoveIsoNode

**ST\_MoveIsoNode** — Moves an isolated node in a topology from one point to another. If new apoint geometry exists as a node an error is thrown. Returns description of move.

#### Synopsis

**text ST\_MoveIsoNode**(varchar atopology, bigint anode, geometry apoint);



XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX. 0 apoint XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
XXXXXX.

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.

XXXXXXXXXXXXXXXXXXXXXXXXXXXX (SRID) XXXXXXXXXXXXXXXXXXXXXXXX.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

Enhanced: 3.2.0 ensures the node cannot be moved in a different face



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.2



```
-- 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
```



## ST AddIsoNode

### 9.6.12 ST NewEdgesSplit

ST\_NewEdgesSplit —  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99\}$ ,  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99\}$ .  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99\}$  ID  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99\}$ .

## Synopsis

bigint **ST\_NewEdgesSplit**(varchar atopology, bigint anedge, geometry apoint);[illegible]

**SRID** (**Spatial Reference ID**) **apoint**, **NULL**, **point**, **geometry**, **geometry**, **geometry**.

Availability: 1.1



This method implements the SOL/MM specification. SOL-MM: Topo-Net Routines: X.3.8



```
-- 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
```



ST\_ModEdgeSplit ST\_ModEdgeHeal ST\_NewEdgeHeal AddEdge

### 9.6.13 ST\_RemoveIsoNode

**ST\_RemoveIsoNode** — Remove an isolated node from the graph. `ST_RemoveIsoNode(g, n)` removes node `n` from the graph `g` if it is isolated (has no incident edges). Returns `1` if the node was removed, `0` otherwise.

## Synopsis

```
text ST_RemoveIsoNode(varchar atopology, bigint anode);
```

[illegible]

Availability: 1.1



✔ This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3



```
-- Remove an isolated node with no face --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7 ) As result;
      result
-----
Isolated node 7 removed
```



## ST AddIsoNode



**Note**

tolerance = 0 `ST_Intersects`, `ST_DWithin`.

GEOS

2.0.0.

`AddEdge`.

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As with1mtol, topology.GetEdgeByPoint(' ←
      ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('SRID=26986;POINT(227622.6 893843)') As geom;
with1mtol | withnotol
-----+-----
          2 |          0
```

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

-- get error --
ERROR:  Two or more edges found
```

`AddEdge`, `GetNodeByPoint`, `GetFaceByPoint`

## 9.7.2 GetFaceByPoint

`GetFaceByPoint` — Finds face intersecting a given point.

### Synopsis

`bigint GetFaceByPoint`(varchar atopology, geometry apoint, float8 tol1);

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.

2.0.0.

Enhanced: 3.2.0 more efficient implementation and clearer contract, stops working with invalid topologies.

❏❏

```
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
```

❏❏

[GetFaceContainingPoint](#), [AddFace](#), [GetNodeByPoint](#), [GetEdgeByPoint](#)

### 9.7.3 GetFaceContainingPoint

**GetFaceContainingPoint** — Finds the face containing a point.

#### Synopsis

bigint **GetFaceContainingPoint**(text atopology, geometry apoint);

❏❏

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

❏❏

[ST\\_GetFaceGeometry](#)

### 9.7.4 GetNodeByPoint

**GetNodeByPoint** — Finds the node-id of a node at a point location.

## Synopsis

bigint **GetNodeByPoint**(varchar atopology, geometry apoint, float8 tol1);

**Notes**

Retrieves the id of a node at a point location.

The function returns an integer (id-node) given a topology, a POINT and a tolerance. If tolerance = 0 means exact intersection, otherwise retrieves the node from an interval.

If apoint doesn't intersect a node, returns 0 (zero).

If use tolerance > 0 and there is more than one node near the point then an exception is thrown.



### Note

tolerance = 0 uses ST\_Intersects, otherwise ST\_DWithin.

GEOS 3.10.0

2.0.0 uses ST\_DWithin.

**Example**

Using **AddEdge** to create a topology.

```
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
```

**See Also**

**AddEdge**, **GetEdgeByPoint**, **GetFaceByPoint**

## 9.7.5 GetTopologyID

GetTopologyID — returns topology ID

## Synopsis

integer **GetTopologyID**(varchar toponame);





Synopsis

varchar **GetTopologyName**(integer topology\_id);

ID topology.topology (integer) .

Availability: 1.1

```
SELECT topology.GetTopologyName(1) As topo_name;
topo_name
-----
ma_topo
```

CreateTopology, DropTopology, GetTopologyID, GetTopologySRID

9.7.8 ST\_GetFaceEdges

ST\_GetFaceEdges — aface (integer) .

Synopsis

getfaceedges\_returntype **ST\_GetFaceEdges**(varchar atopology, bigint aface);

aface (integer) . (sequence) ID(edgeid) (integer) . 1 .

. (integer) .

2.0 .

 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.5

```
-- 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;
```



## GetRingEdges, AddFace, ST GetFaceGeometry

### 9.7.9 ST\_GetFaceGeometry

ST GetFaceGeometry — 16x16x16x16x16x16 ID 16x16x16x16x16x16.

## Synopsis

```
geometry ST_GetFaceGeometry(varchar atopology, bigint aface);
```



XXXXXXXXXXXXXXXX ID XXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.

Availability: 1.1

✔ This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.16



```
-- 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))
```



## AddFace

### 9.7.10 GetRingEdges

**GetRingEdges** — .

## Synopsis

```
getfaceedges_returntype GetRingEdges(varchar atopology, bigint aring, integer max_edges=null);
```



$\text{ID}(\text{edgeid})$ .  $\text{ID}(\text{edgeid})$  (sequence)  $\text{ID}(\text{edgeid})$

[illegible]

```
max_edges = NULL;
// ...
```



### Note

[illegible][illegible]

## ST\_GetFaceEdges, GetNodeEdges

### 9.7.11 GetNodeEdges

GetNodeEdges — .

## Synopsis

```
getfaceedges returntype GetNodeEdges(varchar atopology, bigint anode);
```

[illegible]

## Note

[illegible]

2.0 ☒☒☒☒☒☒☒☒☒☒.



图

图。 **AddEdge** 图, 图, 图。

图, allowEdgeSplitting 图。

computeContainingFace 图。



#### Note

apoint 图, 图 ID(nodeid) 图。

2.0.0 图。

图

```
SELECT topology.AddNode('ma_topo', ST_GeomFromText('POINT(227641.6 893816.5)', 26986) ) As
    nodeid;
-- result --
nodeid
-----
4
```

图

**AddEdge, CreateTopology**

## 9.8.3 AddEdge

AddEdge — 图, 图 (图) 图 ID(edgeid) 图。

### Synopsis

bigint **AddEdge**(varchar toponame, geometry aline);

图

图 toponame 图, 图 (图) 图 ID(edgeid) 图。 图 " (universe)" 图。



#### Note

图 aline 图, 图, 图。

**Note**

aline 的 srid 和 ST\_GeomFromText 的 srid 必须相同。否则将导致不可预料的结果。

GEOS 版本

**Warning**

**AddEdge** is deprecated as of 3.5.0. Use **TopoGeo\_AddLineString** instead.

2.0.0 版本。

SQL

```
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
```

SQL

**TopoGeo\_AddLineString**, **CreateTopology**, Section 4.5

## 9.8.4 AddFace

AddFace — 添加面 (face primitive) 到拓扑。

### Synopsis

bigint **AddFace**(varchar toponame, geometry apolygon, boolean force\_new=false);

面。

面 (face primitive) 面。

面 left\_face 面 right\_face 面。面 containing\_face 面。



#### Note

面 edge 面 next\_left\_edge 面 next\_right\_edge 面。

面 (面) 面。面, 面。

apolygon 面, force\_new 面 (面) 面 ID 面, force\_new 面 ID 面。



#### Note

面 (force\_new = true) 面, 面, 面。面 MBR 面。



#### Note

apolygon 面 srid 面 srid 面。面。

2.0.0 面。

面

```
-- 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
```



## AddEdge, CreateTopology, Section 4.5

### 9.8.5 ST\_Simplify

ST Simplify —  $\text{ST\_Simplify}$ - $\text{ST\_Simplify}$  (Douglas-Peucker)  $\text{ST\_Simplify}$  TopoGeometry  $\text{ST\_Simplify}$   $\text{ST\_Simplify}$   $\text{ST\_Simplify}$ .

## Synopsis

```
geometry ST_Simplify(topogeometry tg, float8 tolerance);
```



**Figure 6:** TopoGeometry “Douglas-Peucker”



### Note

XXXXXXXXXXXXXXXXXXXX, XXXXXXXXXXXXXXXXXXXX.  
 XXXX/XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.

GEOS ☒ ☒ ☒ ☒ ☒

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.



☒☒ ST Simplify, ST IsSimple, ST IsValid, ST ModEdgeSplit

## 9.8.6 RemoveUnusedPrimitives

**RemoveUnusedPrimitives** — Removes topology primitives which not needed to define existing TopoGeometry objects.

### Synopsis

```
bigint RemoveUnusedPrimitives(text topology_name, geometry bbox);
```

¶¶

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

¶¶

[ST\\_ModEdgeHeal](#), [ST\\_RemEdgeModFace](#)

## 9.9 TopoGeometry ¶¶¶

### 9.9.1 CreateTopoGeom

**CreateTopoGeom** — ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. **tg\_type** ¶ 1: [¶¶] ¶¶¶, 2: [¶¶] ¶¶, 3: [¶¶] ¶¶¶, 4: ¶¶¶¶¶¶¶¶.

### Synopsis

```
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id, topoelementarray tg_objs, bigint tg_id);
```

```
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id);
```

¶¶

Creates a topogeometry object for layer denoted by **layer\_id** and registers it in the relations table in the toponame schema.

**tg\_type** is an integer: 1:[multi]point (punctal), 2:[multi]line (lineal), 3:[multi]poly (areal), 4:collection. **layer\_id** is the layer id in the topology.layer table.

¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶, ¶¶, ¶¶, ¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

¶¶¶¶¶¶¶¶¶¶¶¶¶¶ TopoGeometry ¶¶¶¶¶¶¶¶.

Availability: 1.1

例: 建立topogeom

Create a topogeom in ri\_topo schema for layer 2 (our ri\_roads), of type (2) LINE, for the first edge (we loaded in ST\_CreateTopoGeo).

```
INSERT INTO ri.ri_roads(road_name, topo) VALUES('Unknown', topology.CreateTopoGeom('ri_topo' ↵
',2,2,'{1,2}'::topology.topoelementarray);
```

例: 建立topogeom TopoGeometry 例

建立topogeom TopoGeometry。建立 blockgroups 建立 TopoGeometry。建立topogeom TopoGeometry。建立topogeom TopoGeometry。

```
-- 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;
```



```
--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 = toTopoGeom(ST_Buffer(topo, -10), clearTopoGeom(topo), 0);

-- 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
);
```

￼￼

[CreateTopology](#), [AddTopoGeometryColumn](#), [CreateTopoGeom](#), [TopologySummary](#), [clearTopoGeom](#)

### 9.9.3 TopoElementArray\_Agg

**TopoElementArray\_Agg** — Returns a `topoelementarray` for a set of `element_id`, type arrays (`topoelements`).

#### Synopsis

`topoelementarray` **TopoElementArray\_Agg**(`topoelement set tefield`);

￼￼

**TopoElement** ￼￼￼￼￼ **TopoElementArray** ￼￼￼￼￼￼￼￼.

2.0.0 ￼￼￼￼￼￼￼￼￼.

￼￼

```
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}}
```



## TopoElement, TopoElementArray

### 9.9.4 TopoElement

**TopoElement** — Converts a topogeometry to a topoelement.

## Synopsis

```
topoelement TopoElement(topogeometry topo);
```



Converts a **TopoGeometry** to a **TopoElement**.

Availability: 3.4.0



□□□□□□□□□□□□□□□□□□ (workflow) □□□.

```
-- 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;
```



## TopoElementArray Agg, TopoGeometry, TopoElement

## 9.10 TopoGeometry ☒☒☒

### 9.10.1 clearTopoGeom

`clearTopoGeom` — Clears the content of a topo geometry.

## Synopsis

```
topogeometry clearTopoGeom(topogeometry topogeom);
```

图

**TopoGeometry** 函数返回 **TopoGeometry** 类型。 **toTopoGeom** 函数返回 **TopoGeometry** 类型。

2.1 图。

图

```
-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = toTopoGeom(ST_Buffer(topo, -10), clearTopoGeom(topo), 0);
```

图

**toTopoGeom**

### 9.10.2 TopoGeom\_addElement

**TopoGeom\_addElement** — Adds an element to the definition of a TopoGeometry.

#### Synopsis

topogeometry **TopoGeom\_addElement**(topogeometry tg, topoelement el);

图

**TopoGeometry** 函数返回 **TopoElement** 类型。 图。

2.3 图。

图

```
-- Add edge 5 to TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_addElement(tg, '{5,2}');
```

图

**TopoGeom\_remElement, CreateTopoGeom**

### 9.10.3 TopoGeom\_remElement

**TopoGeom\_remElement** — Removes an element from the definition of a TopoGeometry.

#### Synopsis

topogeometry **TopoGeom\_remElement**(topogeometry tg, topoelement el);

❏❏

TopoGeometry ❏❏❏❏❏❏❏ **TopoElement** ❏❏❏❏❏❏❏.

2.3 ❏❏❏❏❏❏❏❏❏❏❏❏❏.

❏❏

```
-- Remove face 43 from TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_remElement(tg, '{43,3}');
```

❏❏

**TopoGeom\_addElement, CreateTopoGeom**

### 9.10.4 TopoGeom\_addTopoGeom

TopoGeom\_addTopoGeom — Adds element of a TopoGeometry to the definition of another TopoGeometry.

#### Synopsis

topogeometry **TopoGeom\_addTopoGeom**(topogeometry tgt, topogeometry src);

❏❏

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

❏❏

```
-- 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
);
```

❏❏

**TopoGeom\_addElement, clearTopoGeom, CreateTopoGeom**



### 9.10.5 toTopoGeom

toTopoGeom — Adds a geometry shape to an existing topo geometry.

文档

Refer to [toTopoGeom](#).

## 9.11 TopoGeometry 文档

### 9.11.1 GetTopoGeomElementArray

GetTopoGeomElementArray — Returns a `topoelementarray` (an array of `topoelements`) containing the topological elements and type of the given `TopoGeometry` (primitive elements).

#### Synopsis

```
topoelementarray GetTopoGeomElementArray(varchar toponame, integer layer_id, bigint tg_id);
topoelementarray GetTopoGeomElementArray(topogeometry tg);
```

文档

`TopoGeometry` 文档 (文档) 返回 `TopoElementArray` 文档。 文档 `GetTopoGeomElements` 文档。

`tg_id` 文档 `topology.layer` 文档 `layer_id` 文档 `TopoGeometry` 文档 `TopoGeometry ID` 文档。

Availability: 1.1

文档

文档

[GetTopoGeomElements](#), [TopoElementArray](#)

### 9.11.2 GetTopoGeomElements

GetTopoGeomElements — Returns a set of `topoelement` objects containing the topological `element_id`, `element_type` of the given `TopoGeometry` (primitive elements).

#### Synopsis

```
setof topoelement GetTopoGeomElements(varchar toponame, integer layer_id, bigint tg_id);
setof topoelement GetTopoGeomElements(topogeometry tg);
```

¶¶

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` ¶ topology.layer ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶ TopoGeometry ¶¶  
 ¶ TopoGeometry ID ¶¶¶.

2.0.0 ¶¶¶¶¶¶¶¶¶¶¶¶.

¶¶

¶¶

**GetTopoGeomElementArray**, **TopoElement**, **TopoGeom\_addElement**, **TopoGeom\_remElement**

### 9.11.3 ST\_SRID

**ST\_SRID** — Returns the spatial reference identifier for a topogeometry.

#### Synopsis

integer **ST\_SRID**(topogeometry tg);

¶¶

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

¶¶

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
-- result
4326
```

¶¶

Section 4.5, **ST\_SetSRID**, **ST\_Transform**, **ST\_SRID**

## 9.12 TopoGeometry 国国国

### 9.12.1 AsGML

AsGML — TopoGeometry 国 GML 国国国国国国国国.

#### Synopsis

```
text AsGML(topogeometry tg);
text AsGML(topogeometry tg, text nsprefix_in);
text AsGML(topogeometry tg, regclass visitedTable);
text AsGML(topogeometry tg, regclass visitedTable, text nsprefix);
text AsGML(topogeometry tg, text nsprefix_in, integer precision, integer options);
text AsGML(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable);
text AsGML(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable,
text idprefix);
text AsGML(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable,
text idprefix, int gmlversion);
```

国国

TopoGeometry 国 GML 国国国 GML3 国国国国国国国国国. nsprefix\_in 国国国国国国国国国 gml 国国国国国国国国. nsprefix 国国国国国国国国国国国国国国 (non-qualified) 国国国国国国国国国国. 国国 (国国 15) 国国 (国国 1) 国国国国国国国国, 国国国国 ST\_AsGML 国国国国国国国国国国国国国国.

visitedTable 国国国国国国国国国国国国国国国国国国国国国国国国, 国国国国, 国国国国国国国国国国国国 (xlink:xref) 国国国国. 国国国国'element\_type' 国'element\_id' (国国) 2 国国国国国国国国. 国国国国国国国国国国国国国国国国国国国国国国国国. 国国国国国国国国, element\_type 国 element\_id 国, 国国国国, 国国国国国国国国国国. 国国国国国国国国国国国国国国国国国国国国国国国国. 国国国国国国国国国国国国国国:

```
CREATE TABLE visited (
  element_type integer, element_id integer,
  unique(element_type, element_id)
);
```

idprefix 国国国国国国国国, 国国国国国国国国国国国国国国国国国国国国国.

gmlver 国国国国国国国国, 国 ST\_AsGML 国国国国国国国国国国国国国国国国. 国国 3 国国. 2.0.0 国国国国国国国国国国.

国国

国国国国 **CreateTopoGeom** 国国国国国国国国国国国国国国.

```
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="-">
```



```

                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>

```

国国

CreateTopoGeom, ST\_CreateTopoGeo

### 9.12.2 AsTopoJSON

AsTopoJSON — TopoGeometry 国 TopoJSON 国国国国国国国国.

#### Synopsis

text **AsTopoJSON**(topogeometry tg, regclass edgeMapTable);

国国

TopoGeometry 国 TopoJSON 国国国国国国国国. edgeMapTable 国 NULL 国国国国, 国国国国国国国国 (arc) 国国国国国国国国/国 (lookup/storage) 国国国国国国国国. 国国国国国国国 (compact) " 国 " 国国国国国国国国国国国国国国国国.

国国国国国国国国, 国国国 " 国 (serial)" 国 "arc id" 国国国国 "edge id" 国国国国国国国国国国. 国国 "edge\_id" 国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国.



#### Note

TopoJSON 国国国国国国国国国国 0-国国国国 "edgeMapTable" 国国国国国 1-国国国国.

国国 TopoJSON 国国, 国国国国国国国国国国 (snippet) 国国, 国国国国国国国国国国国国国国国国国国. **TopoJSON 国国** 国国国国国国.

2.1.0 国国国国国国国国国国.

国国国: 2.2.1 国国国国 (puntal) 国国国国国国国国国国国国.

国国

ST\_AsGeoJSON



Synopsis

boolean **Equals**(topogeometry tg1, topogeometry tg2);

TopoGeometry (, , )



**Note**  
TopoGeometry . TopoGeometry .

1.1.0

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

GetTopoGeomElements, ST\_Equals

9.13.2 Intersects

Intersects — TopoGeometry

Synopsis

boolean **Intersects**(topogeometry tg1, topogeometry tg2);

TopoGeometry



**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).

1.1.0

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

**ST\_Intersects**

## 9.14 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
```

### 9.14.1 Using the Topology exporter

The `pgtopo_export` script takes the name of a database and a topology and outputs a dump file which can be used to import the topology (and associated layers) into a new database.

By default `pgtopo_export` writes the dump file to the standard output so that it can be piped to `pgtopo_import` or redirected to a file (refusing to write to terminal). You can optionally specify an output filename with the `-f` commandline switch.

By default `pgtopo_export` includes a dump of all layers defined against the given topology. This may be more data than you need, or may be non-working (in case your layer tables have complex dependencies) in which case you can request skipping the layers with the `--skip-layers` switch and deal with those separately.

Invoking `pgtopo_export` with the `--help` (or `-h` for short) switch will always print short usage string.

The dump file format is a compressed tar archive of a `pgtopo_export` directory containing at least a `pgtopo_dump_version` file with format version info. As of version 1 the directory contains tab-delimited CSV files with data of the topology primitive tables (node, edge\_data, face, relation), the topology and layer records associated with it and (unless `--skip-layers` is given) a custom-format PostgreSQL dump of tables reported as being layers of the given topology.

### 9.14.2 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` commandline switch.

By default `pgtopo_import` includes in the output SQL file the code to restore all layers found in the dump.

This may be unwanted or non-working in case your target database already have tables with the same name as the ones in the dump. In that case you can request skipping the layers with the `--skip-layers` switch and deal with those separately (or later).

SQL to only load and link layers to a named topology can be generated using the `--only-layers` switch. This can be useful to load layers AFTER resolving the naming conflicts or to link layers to a different topology (say a spatially-simplified version of the starting topology).

If the target topology already exists and you want it dropped upfront you can pass the `--drop-topology` switch (since PostGIS-3.6.0).

# Chapter 10

## 10. Raster Data

### 10.1 raster2pgsql

`raster2pgsql`, a raster loader executable, loads raster data into a PostGIS database.

#### 10.1.1 raster2pgsql

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

`raster2pgsql` (factor) `raster2pgsql`, `raster2pgsql` <http://trac.osgeo.org/postgis/ticket/1764> `raster2pgsql`.

##### 10.1.1.1 Example Usage

`raster2pgsql` 100x100 `raster2pgsql`:

```
# -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.

UNIX 安装和配置指南:

```
raster2pgsql -s 4326 -I -C -M *.tif -F -t 100x100 public.demelevation | psql -d gisdb
```

安装和配置指南: `aerial` 数据库, 使用 2 到 4 个数据库连接, 数据库 (数据库 DB 名称) 名称, 数据库名称和数据库名称 -e 数据库名称 (数据库名称和数据库名称). 使用 128x128 数据库名称. 数据库名称. 数据库名称. 数据库名称 -F 数据库名称 "filename" 数据库名称.

```
raster2pgsql -I -C -e -Y -F -s 26986 -t 128x128 -l 2,4 bostonaerials2008/*.jpg aerals. ↵
boston | psql -U postgres -d gisdb -h localhost -p 5432
```

--get a list of raster types supported:

```
raster2pgsql -G
```

-G 安装和配置指南:

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
```

### 10.1.1.2 raster2pgsql options

-? 安装和配置指南. 安装和配置指南.

-G 安装和配置指南.

**c|a|d|p** -- 安装和配置指南:

- c 安装和配置指南 (X) 安装和配置指南. 安装和配置指南.
- a 安装和配置指南 (X) 安装和配置指南.
- d 安装和配置指南, 安装和配置指南 (X) 安装和配置指南.
- p 安装和配置指南, 安装和配置指南.

安装和配置指南: 安装和配置指南

- C `raster_columns` 安装和配置指南 SRID, 安装和配置指南.
- x 安装和配置指南 (extent) 安装和配置指南. -C 安装和配置指南.

**-r** **regular blocking** 使用正则阻塞 (regular blocking) 模式。-C 选项  
指定正则阻塞的块大小。

**选项:** 指定正则阻塞的块大小。

**-s <SRID>** 指定 SRID 值。如果为 0，则使用 SRID 值。  
指定正则阻塞的块大小。

**-b BAND** 指定 (1-255) 的波段。指定正则阻塞的块大小，  
指定正则阻塞的块大小。

**-t TILE\_SIZE** 指定 TILE\_SIZE 值。TILE\_SIZE 指定 x 指定  
指定正则阻塞的块大小，指定正则阻塞的块大小。

**-P** 指定正则阻塞的块大小 (padding) 指定正则阻塞的块大小。

**-R, --register** 指定正则阻塞的块大小 (DB 指定正则阻塞的块大小)。  
指定正则阻塞的块大小 (指定正则阻塞的块大小)。

**-l OVERVIEW\_FACTOR** 指定正则阻塞的块大小。指定正则阻塞的块大小，  
指定正则阻塞的块大小 o overview\_factor\_table 指定正则阻塞的块大小，  
指定正则阻塞的块大小 overview\_factor 指定正则阻塞的块大小。  
指定正则阻塞的块大小 (placeholder) 指定正则阻塞的块大小 table 指定正则阻塞的块大小。  
指定正则阻塞的块大小，-R 指定正则阻塞的块大小。指定正则阻塞的块大小 SQL 指定正则阻塞的块大小。  
指定正则阻塞的块大小。

**-N NODATA** "NODATA" 指定正则阻塞的块大小 NODATA 指定正则阻塞的块大小。

**选项:** 指定正则阻塞的块大小。

**-f COLUMN** 指定正则阻塞的块大小。指定正则阻塞的块大小 'rast' 指定正则阻塞的块大小。

**-F** 指定正则阻塞的块大小。

**-n COLUMN** 指定正则阻塞的块大小。-F 指定正则阻塞的块大小。

**-q** PostgreSQL 指定正则阻塞的块大小。

**-I** 指定正则阻塞的块大小 GiST 指定正则阻塞的块大小。

**-M** 指定正则阻塞的块大小 (vacuum analyze) 指定正则阻塞的块大小。

**-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** 指定正则阻塞的块大小。-X 指定正则阻塞的块大小 (指定正则阻塞的块大小)  
指定正则阻塞的块大小。

**-X tablespace** 指定正则阻塞的块大小。-I 指定正则阻塞的块大小。  
指定正则阻塞的块大小。

**-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** 指定正则阻塞的块大小，指定正则阻塞的块大小 (transaction) 指定正则阻塞的块大小。

**-E ENDIAN** 指定正则阻塞的块大小 (endianness) 指定正则阻塞的块大小。XDR 指定正则阻塞的块大小 0，指定正则阻塞的块大小  
指定正则阻塞的块大小 1 指定正则阻塞的块大小。指定正则阻塞的块大小，NDR 指定正则阻塞的块大小。

**-V version** 指定正则阻塞的块大小。指定正则阻塞的块大小 0 指定正则阻塞的块大小。指定正则阻塞的块大小，0 指定正则阻塞的块大小。

### 10.1.2 PostGIS 简体中文版

指定正则阻塞的块大小。指定正则阻塞的块大小，指定正则阻塞的块大小。  
指定正则阻塞的块大小。指定正则阻塞的块大小。

1. 指定正则阻塞的块大小:

```
CREATE TABLE myrasters(rid serial primary key, rast raster);
```

2. `ST_MakeEmptyRaster`, `ST_AddBand`, `ST_AsRaster`, `ST_Union`, `ST_MapAlgebra` (algebra) `ST_Transform`.
3. `ST_ConvexHull`:

```
CREATE INDEX myrasters_rast_st_convexhull_idx ON myrasters USING gist( ST_ConvexHull(
rast) );
```

(convex hull) `ST_ConvexHull`.



#### Note

PostGIS 2.0 `envelop` `ST_ConvexHull`.

4. `AddRasterConstraints`

### 10.1.3 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=xxxxxxxxxxxxxxxxxxxxx \
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \
raster2pgsql \
-s 990000 \
-t 256x256 \
-I \
-R \
/vs3/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_vsi_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.

## 10.2 Raster Cloud

PostGIS `raster` can interact with cloud storage providers like Amazon S3, Google Cloud Storage, and Microsoft Azure Blob Storage. This section describes how to configure PostGIS to use these providers for storing and retrieving raster data.

1. `raster_columns` table: This table stores the metadata for each raster column in the database. It is located in the `public` schema.
2. `raster_overviews` table: This table stores the metadata for each raster overview. It is located in the `public` schema. The `-l` option in the `raster2pgsql` command is used to generate overviews.

### 10.2.1 Raster Cloud Configuration

`raster_columns` table: This table stores the metadata for each raster column in the database. It is located in the `public` schema. The `raster_columns` table has the following columns: `r_table_catalog`, `r_table_schema`, `r_table_name`, `r_raster_column`, `srid`, `scale_x`, `scale_y`, `add_raster_constraints`, and `add_raster_constraints_sql`.

`raster_overviews` table: This table stores the metadata for each raster overview. It is located in the `public` schema. The `raster_overviews` table has the following columns: `r_table_catalog`, `r_table_schema`, `r_table_name`, `r_raster_column`, `srid`, `scale_x`, `scale_y`, `add_raster_constraints`, and `add_raster_constraints_sql`.

- `r_table_catalog`: The catalog name of the table. It is `public` by default.
- `r_table_schema`: The schema name of the table. It is `public` by default.
- `r_table_name`: The table name of the raster column.
- `r_raster_column`: The name of the raster column in the table. PostGIS uses this column to store the raster data. The `raster_columns` table has a `check` constraint that ensures the `r_raster_column` is not null.
- `srid`: The spatial reference ID of the raster. Section 4.5 describes how to set the `srid`.
- `scale_x`: The scale factor of the raster in the X direction. The `scale_x` is `1` by default. The `scale_x` can be set to a value other than `1` if the raster is not in the standard projection. The `scale_x` is `1` by default.

- `scale_y` 縦方向のスケール (倍率) を指定。縦方向のスケールは `scale_y` で指定。横方向のスケールは `ST_ScaleX` で指定。
- `blocksize_x` 横方向のブロックサイズ (ピクセル) を指定。横方向のブロックサイズは `ST_Width` で指定。
- `blocksize_y` 縦方向のブロックサイズ (ピクセル) を指定。縦方向のブロックサイズは `ST_Height` で指定。
- `same_alignment` 出力の出力位置を指定。出力位置は `ST_SameAlignment` で指定。
- `regular_blocking` 規則的なブロック化を指定。規則的なブロック化は `ST_RegularBlocking` で指定。
- `num_bands` 出力の出力帯域数を指定。出力帯域数は `ST_NumBands` で指定。
- `pixel_types` 出力の出力ピクセルタイプを指定。出力ピクセルタイプは `ST_BandPixelType` で指定。
- `nodata_values` 出力の出力ノodata値を指定 (double precision) の数 (数) を指定。出力ノodata値は `ST_BandNoDataValue` で指定。
- `out_db` 出力の出力データベースを指定。出力データベースは `ST_OutDB` で指定。
- `extent` 出力の出力範囲 (extent) を指定。出力範囲は `DropRasterConstraints` で指定。出力範囲は `AddRasterConstraints` で指定。
- `spatial_index` 出力の出力空間インデックスを指定。出力空間インデックスは `ST_SpatialIndex` で指定。

### 10.2.2 ☐ ☐ ☐ ☐ ☐ ☐

```
raster_overviews --help raster_columns raster_overviews --help.
--help. -l --help. AddOverviewConstraints
```

☐.



### Note

```
raster_overviews || raster_columns || raster_columns
|| raster_overviews || raster_columns ||
|| (join) ||
```

[illegible]

- [illegible]

raster overviews .





```

        $input_srid = intval($_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)]),
        ,3)))
    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]);
?>

```

### 10.3.2 使用 ST\_AsPNG 在 ASP.NET C# 中显示栅格

在 npgsql PostgreSQL .NET 中使用 **ST\_AsGDALRaster** 函数可以返回 1, 2, 3 个 PHP 请求流 (request stream) 的栅格数据。PHP 使用 `img src` HTML 属性来显示栅格。

在 npgsql PostgreSQL .NET 中，请参见 <http://npgsql.projects.postgresql.org/> 获取更多信息。在 ASP.NET bin 目录中，请参见 `ST_AsPNG.dll`。

在 npgsql PostgreSQL .NET 中，`ST_Union` (combine) 函数将 WGS84 坐标的栅格数据合并，`ST_Transform` (union) 函数将栅格数据转换为指定的坐标系，`ST_AsPNG` 函数将栅格数据转换为 PNG 格式。

在 C# 中，请参见 Section 10.3.1 中的示例。

[http://mywebserver/test\\_raster.php?srid=2249](http://mywebserver/test_raster.php?srid=2249)

在 web.config 文件中，请添加以下配置。

```

-- 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;
}

```

### 10.3.3 透過 Npgsql 在 Java 中執行 PostGIS 查詢

以下範例展示了如何在 Java 中使用 Npgsql 執行 PostGIS 查詢。

<http://jdbc.postgresql.org/download.html> 提供了 PostgreSQL JDBC 驅動程式的下載資訊。  
 有關 Npgsql 的詳細資訊，請參閱：  
<http://npgsql.org/>

```
set env CLASSPATH ../../postgresql-9.0-801.jdbc4.jar
javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class
```

執行指令如下：

```
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);
    }
}
}

```

### 10.3.4 PLPython 與 SQL 的整合

PostgreSQL 提供了 PLPython 這個函式語言。PLPython 是 PostgreSQL 的 PLPythonu 與 PLPythonu3u 的整合。

```

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

```

### 10.3.5 PSQL 的 lo 函式

PostgreSQL 提供了 PSQL 的 lo 函式。PostgreSQL 的 lo 函式是 PostgreSQL 的 PSQL 的 lo 函式。PSQL 的 lo 函式，是 PostgreSQL 的 PSQL 的 lo 函式。

PostgreSQL 的 lo 函式，是 PostgreSQL 的 PSQL 的 lo 函式。

```

SELECT oid, lowrite(lo_open(oid, 131072), png) As num_bytes
FROM
( VALUES (lo_create(0),
    ST_AsPNG( (SELECT rast FROM aerials.boston WHERE rid=1) )
) ) As v(oid,png);
-- you'll get an output something like --
oid | num_bytes
-----+-----
2630819 | 74860

```

```
-- next note the oid and do this replacing the c:/test.png to file path location
-- on your local computer
\lo_export 2630819 'C:/temp/aerial_samp.png'

-- this deletes the file from large object storage on db
SELECT lo_unlink(2630819);
```



## 11.1 几何函数

### 11.1.1 geomval

**geomval** — (geometry) geom 的 (geometry) val, 返回 geometry 值。

例

**geomval** geometry, .geom geometry 返回 geometry 值。 **ST\_DumpAsPolygon** 返回 geometry 值。

例

Section 13.6

### 11.1.2 addbandarg

**addbandarg** — 返回 ST\_AddBand 返回的 geometry 值。

例

返回 ST\_AddBand 返回的 geometry 值。

**index integer** 返回 1-geometry。 NULL 返回, 返回 geometry。

**pixeltype text** **ST\_BandPixelType** 返回 geometry。

**initialvalue double precision** 返回 geometry。

**nodataval double precision** 返回 NODATA 值。 NULL 返回, 返回 NODATA 值。

例

**ST\_AddBand**

### 11.1.3 rastbandarg

**rastbandarg** — 返回 geometry 值。

例

返回 geometry 值。

**rast raster** 返回 geometry。

**nband integer** 返回 1-geometry。

ST\_MapAlgebra (callback function version)

11.1.4 raster

raster —

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**  
GDAL PostGIS. ST\_ConvexHull.

|  |  |
|--|--|
|  |  |
|  |  |

Chapter 11

11.1.5 reclassarg

reclassarg — A composite type used as input into the ST\_Reclass function defining the behavior of reclassification.

A composite type used as input into the ST\_Reclass function defining the behavior of reclassification.

nband integer

reclassexpr text range:map\_range

- 1. [a-b] = a <= x <= b
- 2. (a-b) = a < x <= b



3.  $[a-b] = a \leq x < b$

4.  $(a-b) = a < x < b$

'(  $[a-b]$  )  $a-b$  。

**pixeltype text** **ST\_BandPixelType** 。

**nodataval double precision** NODATA 。

： 2 255 NODATA 8BUI 。

```
SELECT ROW(2, '0-100:1-10, 101-500:11-150,501 - 10000: 151-254', '8BUI', 255)::reclassarg;
```

： 1 NODATA 1BB 。

```
SELECT ROW(1, '0-100]:0, (100-255:1', '1BB', NULL)::reclassarg;
```

。

**ST\_Reclass**

### 11.1.6 summarystats

summarystats — ST\_SummaryStats 或 ST\_SummaryStatsAgg 。

。

**ST\_SummaryStats** 或 **ST\_SummaryStatsAgg** 。

**count integer** 。

**sum double precision** 。

**mean double precision** 。

**stddev double precision** 。

**min double precision** 。

**max double precision** 。

。

**ST\_SummaryStats**, **ST\_SummaryStatsAgg**

### 11.1.7 unionarg

unionarg — UNION 或 ST\_Union 。

例

```
CREATE TABLE test UNION SELECT ST_Union (geom) FROM test;
```

**nband integer** 指定栅格表的带数，范围 1-255。

**uniontype text** UNION 操作的结果类型。 **ST\_Union** 操作的结果类型。

例

**ST\_Union**

## 11.2 栅格函数

### 11.2.1 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 no-
data_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[] VARI-
ADIC 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 );
```

例

```
raster_columns 列出所有栅格表，并返回其约束信息。
rastschema 指定栅格表的模式名。 srid 指定 SPATIAL_REF_SYS 表中的 SRID。
```

```
raster2pgsql 将栅格数据加载到数据库中。
```

有关栅格约束的详细信息，请参见 Section 10.2.1 栅格约束。

- **blocksize** X Y 指定栅格表的块大小。
- **blocksize\_x** X 指定 (X 方向) 的块大小。
- **blocksize\_y** Y 指定 (Y 方向) 的块大小。
- **extent** 指定栅格表的范围。指定栅格表的范围。
- **num\_bands** 指定栅格表的带数。

- `pixel_types` 儲存每個像素的類型。每個 N 儲存每個像素的類型。
- `regular_blocking` 儲存每個像素的類型 (儲存每個像素的類型) 儲存每個像素的類型 (儲存每個像素的類型) 儲存每個像素的類型。
- `same_alignment` ensures they all have same alignment meaning any two tiles you compare will return true for. Refer to [ST\\_SameAlignment](#).
- `srid` 儲存每個像素的 SRID 儲存每個像素的 SRID。
- `--` 儲存每個像素的 SRID 儲存每個像素的 SRID。

**Note**

儲存每個像素的 SRID 儲存每個像素的 SRID。儲存每個像素的 SRID, 儲存每個像素的 SRID。

**Note**

儲存每個像素的 SRID 儲存每個像素的 SRID。儲存每個像素的 SRID, 儲存每個像素的 SRID。

## 2.0.0 儲存每個像素的 SRID。

儲存每個像素的 SRID: 儲存每個像素的 SRID。

```
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}
```

儲存每個像素的 SRID: 儲存每個像素的 SRID。

```
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
```

❏

Section [10.2.1, ST\\_AddBand, ST\\_MakeEmptyRaster, DropRasterConstraints, ST\\_BandPixelType, ST\\_SRID](#)

## 11.2.2 DropRasterConstraints

**DropRasterConstraints** — 删除 PostGIS 中的栅格约束。该函数将指定的栅格表中的指定列上的所有约束删除。

### 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);

❏

**AddRasterConstraints** 函数，将指定的栅格表中的指定列上的所有约束添加到指定的栅格表中。该函数将指定的栅格表中的指定列上的所有约束添加到指定的栅格表中。

该函数将指定的栅格表中的指定列上的所有约束添加到指定的栅格表中。

```
DROP TABLE mytable
```

该函数将指定的栅格表中的指定列上的所有约束添加到指定的栅格表中，SQL 语句如下。

```
ALTER TABLE mytable DROP COLUMN rast
```

该函数将指定的栅格表中的指定列上的所有约束添加到指定的栅格表中。该函数将指定的栅格表中的指定列上的所有约束添加到指定的栅格表中。

2.0.0 版本。



```
SELECT DropRasterConstraints ('myrasters','rast');
----RESULT output ---
t

-- verify change in raster_columns --
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types,
nodata_values
FROM raster_columns
WHERE r_table_name = 'myrasters';

srid | scale_x | scale_y | blocksize_x | blocksize_y | num_bands | pixel_types |
nodata_values
-----+-----+-----+-----+-----+-----+-----+-----
0 | | | | | | |
```

```

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)

```



Section 10.2.2, DropOverviewConstraints, ST\_CreateOverview, AddRasterConstraints

### 11.2.4 DropOverviewConstraints

DropOverviewConstraints —  (overview) .

## Synopsis

```
boolean DropOverviewConstraints(name ovschema, name ovtbl, name ovcolumn);
boolean DropOverviewConstraints(name ovtbl, name ovcolumn);
```



```
raster_overviews ██████████
██████.
```

```
ovschema [REDACTED], search path [REDACTED].
```

[illegible]

## Section 10.2.2, AddOverviewConstraints, DropRasterConstraints

### 11.2.5 PostGIS\_GDAL\_Version

PostGIS GDAL Version — PostGIS 2.5.3 GDAL 2.2.4.

## Synopsis

```
text PostGIS_GDAL_Version();
```

❏

PostGIS 简体中文 GDAL 简体中文. ❏ GDAL 简体中文. ❏.

❏

```
SELECT PostGIS_GDAL_Version();
       postgis_gdal_version
-----
GDAL 1.11dev, released 2013/04/13
```

❏

[postgis.gdal\\_datapath](#)

### 11.2.6 PostGIS\_Raster\_Lib\_Build\_Date

PostGIS\_Raster\_Lib\_Build\_Date — 简体中文.

#### Synopsis

text **PostGIS\_Raster\_Lib\_Build\_Date()**;

❏

简体中文.

❏

```
SELECT PostGIS_Raster_Lib_Build_Date();
       postgis_raster_lib_build_date
-----
2010-04-28 21:15:10
```

❏

[PostGIS\\_Raster\\_Lib\\_Version](#)

### 11.2.7 PostGIS\_Raster\_Lib\_Version

PostGIS\_Raster\_Lib\_Version — 简体中文.

#### Synopsis

text **PostGIS\_Raster\_Lib\_Version()**;

---

```
SELECT PostGIS_Raster_Lib_Version();
postgis_raster_lib_version
-----
2.0.0
```

PostGIS\_Lib\_Version

11.2.8 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);

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.

2.0.6, 2.1.3 - GUC gdal\_enabled\_drivers,

2.0.0. GDAL 1.6.0.

:

```
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:** `SELECT create_options FROM st_gdaldrivers()`

```
-- 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                                       | ←                             |
| 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)



[ST\\_AsGDALRaster](#), [ST\\_SRID](#), [postgis.gdal\\_enabled\\_drivers](#)

## 11.2.9 UpdateRasterSRID

UpdateRasterSRID — 更改栅格表列的 SRID 值。

### 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);
```

更新

将指定栅格表列的 SRID 值更改为 new\_srid。如果 new\_srid 为 0，则将 SRID 值更改为栅格数据的默认 SRID (4326)。



#### Note

更新操作是原子性的。如果失败，则不会更改任何数据。

2.1.0 版本引入。

更新

## UpdateGeometrySRID

## 11.2.10 ST\_CreateOverview

ST\_CreateOverview — 创建栅格表的概述图。

### Synopsis

```
regclass ST_CreateOverview(regclass tab, name col, int factor, text algo='NearestNeighbor');
```

概述

创建指定栅格表列的概述图。概述图是原始栅格数据的缩小版本，其分辨率由 factor 参数控制。

概述图存储在 raster\_overviews 表中。

支持的 resampling methods 包括 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', 'Lanczos'。有关更多 resampling methods 的详细信息，请参阅 [GDAL Warp resampling methods](#)。

2.2.0 版本引入。

更新

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);
```



ST\_Retile, AddOverviewConstraints, AddRasterConstraints, Section 10.2.2

### 11.3 (constructor)

### 11.3.1 ST\_AddBand

ST AddBand —       ( ) .   ,   .

## 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);



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.

**addbandarg**  1  **addbandarg**  **addbandarg**   
     .

XXXXXXXXXXXXXXXXXXXX 5 XXXX, torast X NULL XXXXXXXXXXXXXXXXXXXX fromband XXXXXXXX  
XXXXXX (累計) XXXX.

outdbfile 6 7 , outdbfile .  
PostgreSQL .

2.1.0 addbandarg 2.1.0.

MySQL: 2.1.0 DB MySQL.

☐ ☐ : ☐ ☐ ☐ ☐ ☐ ☐

```
-- 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
```

Example: ST\_AddBand

```
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;
```

**DB**

```
SELECT
    *
FROM ST_BandMetadata(
    ST_AddBand(
        ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
        '/home/raster/mytestraster.tif'::text, NULL::int[]
    ),
    ARRAY[]::integer[]
);
```

| bandnum | pixeltype | nodatavalue | isoutdb | path                          |
|---------|-----------|-------------|---------|-------------------------------|
| 1       | 8BUI      |             | t       | /home/raster/mytestraster.tif |
| 2       | 8BUI      |             | t       | /home/raster/mytestraster.tif |
| 3       | 8BUI      |             | t       | /home/raster/mytestraster.tif |

[ST\\_BandMetaData](#), [ST\\_BandPixelType](#), [ST\\_MakeEmptyRaster](#), [ST\\_MetaData](#), [ST\\_NumBands](#), [ST\\_Reclass](#)

### 11.3.2 ST\_AsRaster

ST\_AsRaster — PostGIS PostGIS

#### Synopsis

raster **ST\_AsRaster**(geometry geom, raster ref, text pixeltype, double precision value=1, double precision nodataval=0, boolean touched=false);

raster **ST\_AsRaster**(geometry geom, raster ref, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], boolean touched=false);

raster **ST\_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST\_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);





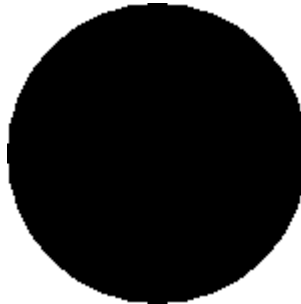
2.0.0 00000000000000000000. GDAL 1.6.0 00000000000000000000.



### Note

XXXXXXXXXX, TIN, XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX, GDAL XXXXXXXXXXXXXXXXXXXX  
XXXXXXXXXXXXXXXXXX.

☒☒: **PNG** ☒☒☒☒☒☒☒☒



```
-- 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));
```



*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]));
```



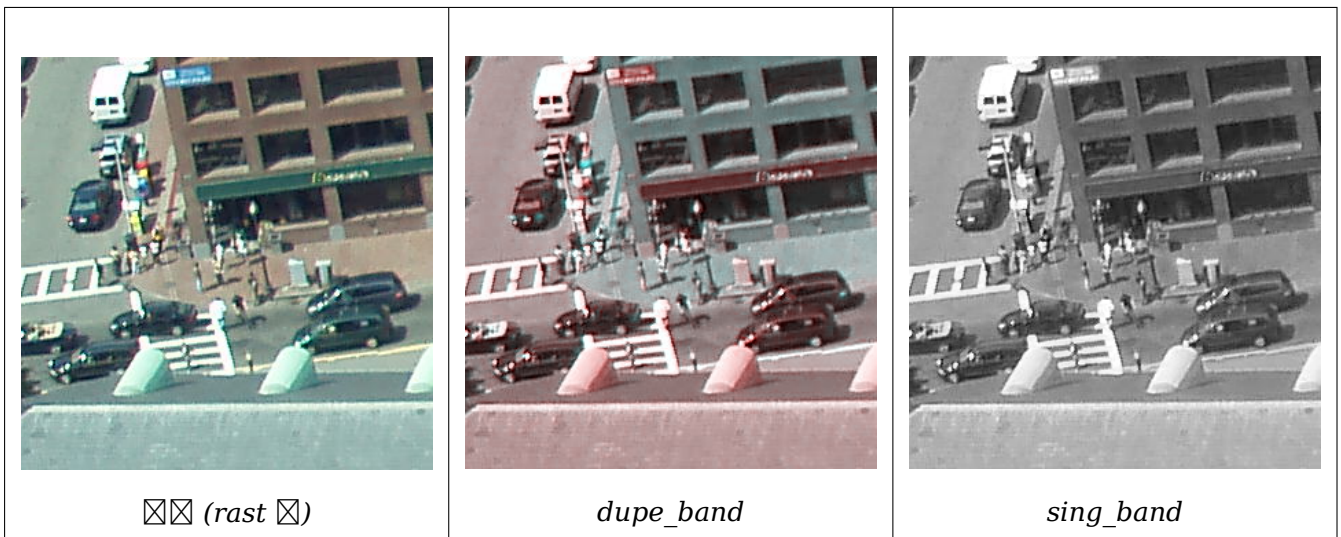
ST BandPixelType, ST Buffer, ST GDALDrivers, ST AsGDALRaster, ST AsPNG, ST AsJPEG, ST SRID

### 11.3.3 ST\_AsRasterAgg

ST AsRasterAgg — Aggregate. Renders PostGIS geometries into a new raster.







```
--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;
```

[ST\\_AddBand](#), [ST\\_NumBands](#), [ST\\_Reclass](#), Chapter 11

### 11.3.5 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);

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.

2.2.0

Examples

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<br>numbands | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  |
|------------------------|------------|-------|--------|--------|--------|-------|-------|------|--|
| 22                     | 33         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 30.25                  | 0          | 33    | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 38.5                   | 0          | 33    | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 46.75                  | 0          | 33    | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 22                     | 22         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 30.25                  | 0          | 22    | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 38.5                   | 0          | 22    | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 46.75                  | 0          | 22    | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 22                     | 11         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 30.25                  | 0          | 11    | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 38.5                   | 0          | 11    | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 46.75                  | 0          | 11    | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 22                     | 0          | 0     | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 30.25                  | 0          | 0     | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 38.5                   | 0          | 0     | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |
| 46.75                  | 0          | 0     | 1      | 8.25   | -11    | 0     | 0     | 4326 |  |

Notes

ST\_MakeEmptyRaster

11.3.6 ST\_MakeEmptyRaster

ST\_MakeEmptyRaster — Create an empty raster (width & height), width X height, scale (scalex, scaley, skewx & skewy) and SRID (SRID) raster. The SRID is optional. If not specified, the SRID is 0(unknown).

## 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);
```



00000 (00 & 00), 00 (0000) 000000000000 X(upperleftx) 0000 Y(upperlefty), 00  
 000, 00 (scalex, scaley, skewx & skewy) 000000000000 (SRID) 00000 (00000) 000  
 0000000.

00000000 (pixelsize) 00000000000000000000. scalex 00000000,  
 scaley 0000000000000000. skewx 0 skewy 0 00000000.

□□□□□□□□□□, □□□□□□□□□□□□□□□□ (□□□□□) □□□□□□□□□□.

SRID 0. 0. ST AddBand, ST SetValue.



```
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 |    0 | ↵
    4326 |          | 0
  4 |    0.0005 |    0.0005 |   100 |   100 |      1 |      1 |     0 |     0 |    0 | ↵
    4326 |          | 0
```



ST AddBand, ST MetaData, ST ScaleX, ST ScaleY, ST SetValue, ST SkewX, , ST SkewY

### 11.3.7 ST Tile

ST Tile — 

## Synopsis

```
setof raster ST_Tile(raster rast, int[] nband, integer width, integer height, boolean padwithnodata=FALSE,
double precision nodataval=NULL);
setof raster ST_Tile(raster rast, integer nband, integer width, integer height, boolean padwithno-
data=FALSE, double precision nodataval=NULL);
setof raster ST_Tile(raster rast, integer width, integer height, boolean padwithnodata=FALSE, dou-
ble precision nodataval=NULL);
```

[illegible]

```
padwithnodata = FALSE ##, #####  
#####.  
padwithnodata = TRUE ##, #####  
NODATA #### (padding) #####. ##### ( ) ##### NODATA ( ) #####  
###, nodataval ##### NODATA #####.
```



### Note

DB , DB .

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.



```

WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', 1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', 2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', 3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
    SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
    SELECT ST_Tile(rast, 3, 3, TRUE) AS rast FROM bar
)
SELECT
    ST_DumpValues(rast)
FROM baz;

```



## st\_dumpvalues

```

-----
(1,"{{1,1,1},{1,1,1},{1,1,1}}")
(2,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{2,2,2},{2,2,2},{2,2,2}}")
(2,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{3,3,3},{3,3,3},{3,3,3}}")
(2,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{4,4,4},{4,4,4},{4,4,4}}")
(2,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{5,5,5},{5,5,5},{5,5,5}}")
(2,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{6,6,6},{6,6,6},{6,6,6}}")
(2,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{7,7,7},{7,7,7},{7,7,7}}")
(2,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{8,8,8},{8,8,8},{8,8,8}}")
(2,"{{80,80,80},{80,80,80},{80,80,80}}")
(1,"{{9,9,9},{9,9,9},{9,9,9}}")
(2,"{{90,90,90},{90,90,90},{90,90,90}}")
(18 rows)

```

```

WITH foo AS (
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
    1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
    2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
    3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
    ', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
    ', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
    ', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
    ', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
    ', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
    ', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
  SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
  SELECT ST_Tile(rast, 3, 3, 2) AS rast FROM bar
)
SELECT
  ST_DumpValues(rast)
FROM baz;

```

## st\_dumpvalues

```

-----
(1,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{80,80,80},{80,80,80},{80,80,80}}")

```

```
(1,"{{90,90,90},{90,90,90},{90,90,90}}")
(9 rows)
```

文档

[ST\\_Union](#), [ST\\_Retile](#)

### 11.3.8 ST\_Retile

`ST_Retile` — 将栅格重采样为指定的宽度和高度。

#### Synopsis

SETOF raster **ST\_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

文档

`sfx`, `sfy` 指定栅格的缩放比例，`tw`, `th` 指定栅格的宽度和高度，`tab`, `col` 指定栅格的表和列，`ext` 指定栅格的几何类型。

默认值为 'NearestNeighbor'，其他可选值为 'Bilinear'，'Cubic'，'CubicSpline'，以及 'Lanczos'。详细请参见 [GDAL Warp resampling methods](#)。

2.2.0 版本引入。

文档

[ST\\_CreateOverview](#)

### 11.3.9 ST\_FromGDALRaster

`ST_FromGDALRaster` — 从 GDAL 数据集创建栅格。

#### Synopsis

raster **ST\_FromGDALRaster**(bytea gdaldata, integer srid=NULL);

文档

`gdaldata` 是 GDAL 数据集的字节数据，`gdaldata` 是 bytea 类型的 GDAL 数据集的字节数据。

`srid` 是 SRID，默认为 NULL。如果 `srid` 不为 NULL，则返回的栅格将具有指定的 SRID。详细请参见 [SRID](#)。

2.1.0 版本引入。

```
WITH foo AS (  
    SELECT ST_AsPNG(ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.1, ←  
        -0.1, 0, 0, 4326), 1, '8BUI', 1, 0), 2, '8BUI', 2, 0), 3, '8BUI', 3, 0)) AS png  
) ,  
bar AS (  
    SELECT 1 AS rid, ST_FromGDALRaster(png) AS rast FROM foo  
    UNION ALL  
    SELECT 2 AS rid, ST_FromGDALRaster(png, 3310) AS rast FROM foo  
)  
SELECT  
    rid,  
    ST_Metadata(rast) AS metadata,  
    ST_SummaryStats(rast, 1) AS stats1,  
    ST_SummaryStats(rast, 2) AS stats2,  
    ST_SummaryStats(rast, 3) AS stats3  
FROM bar  
ORDER BY rid;
```

| rid | metadata                  | stats1        | stats2        | stats3         |
|-----|---------------------------|---------------|---------------|----------------|
| 1   | (0,0,2,2,1,-1,0,0,0,3)    | (4,4,1,0,1,1) | (4,8,2,0,2,2) | (4,12,3,0,3,3) |
| 2   | (0,0,2,2,1,-1,0,0,3310,3) | (4,4,1,0,1,1) | (4,8,2,0,2,2) | (4,12,3,0,3,3) |

(2 rows)

ST\_AsGDALRaster

11.4 (accessor)

11.4.1 ST\_GeoReference

ST\_GeoReference — (world) GDAL ESRI . GDAL .

Synopsis

text ST\_GeoReference(raster rast, text format=GDAL);

(carriage) GDAL ESRI . GDAL . 'GDAL' 'ESRI' .

:

GDAL:

scalex  
skewy  
skewx  
scaley

upperleftx  
upperlefty

ESRI:

scalex  
skewy  
skewx  
scaley  
upperleftx + scalex\*0.5  
upperlefty + scaley\*0.5

```
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 |

**ST\_SetGeoReference, ST\_ScaleX, ST\_ScaleY**

11.4.2 ST\_Height

ST\_Height —

Synopsis

integer **ST\_Height**(raster rast);

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;
```

| rid | rastheight |
|-----|------------|
| 1   | 20         |
| 2   | 5          |

函数

**ST\_Width**

### 11.4.3 ST\_IsEmpty

**ST\_IsEmpty** — 检查空几何体 (width = 0, height = 0) 是否有效。返回布尔值。

#### Synopsis

boolean **ST\_IsEmpty**(raster rast);

函数

检查空几何体 (width = 0, height = 0) 是否有效。返回布尔值。  
2.0.0 版本引入。

函数

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0, 0))
st_isempty |
-----+
f          |

SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0, 0))
st_isempty |
-----+
t          |
```

函数

**ST\_HasNoBand**

### 11.4.4 ST\_MemSize

**ST\_MemSize** — 返回几何体 (几何体) 的内存大小。

#### Synopsis

integer **ST\_MemSize**(raster rast);



|   |  |            |  |         |  |    |  |    |  |      |  |       |  |   |  |   |  |   |  |   |
|---|--|------------|--|---------|--|----|--|----|--|------|--|-------|--|---|--|---|--|---|--|---|
| 1 |  | 0.5        |  | 0.5     |  | 10 |  | 20 |  | 2    |  | 3     |  | 0 |  | 0 |  | 0 |  | ↵ |
| 2 |  | 3427927.75 |  | 5793244 |  | 5  |  | 5  |  | 0.05 |  | -0.05 |  | 0 |  | 0 |  | 0 |  | ↵ |
|   |  |            |  |         |  |    |  |    |  |      |  |       |  |   |  |   |  |   |  |   |
|   |  | 0          |  |         |  |    |  |    |  |      |  |       |  |   |  |   |  |   |  |   |
|   |  | 3          |  |         |  |    |  |    |  |      |  |       |  |   |  |   |  |   |  |   |

**ST\_BandMetaData**, **ST\_NumBands**

11.4.6 ST\_NumBands

ST\_NumBands — Returns the number of bands in a raster.

Synopsis

integer **ST\_NumBands**(raster rast);

```
SELECT rid, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

| rid | numbands |
|-----|----------|
| 1   | 0        |
| 2   | 3        |

**ST\_Value**

11.4.7 ST\_PixelHeight

ST\_PixelHeight — Returns the pixel height of a raster.

Synopsis

double precision **ST\_PixelHeight**(raster rast);

**ST\_PixelWidth**

**Example:**

```
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     |

**Example: 0 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 (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   |

**Example:**

**ST\_PixelWidth, ST\_ScaleX, ST\_ScaleY, ST\_SkewX, ST\_SkewY**

**11.4.8 ST\_PixelWidth**

ST\_PixelWidth — Returns the pixel width of a raster.

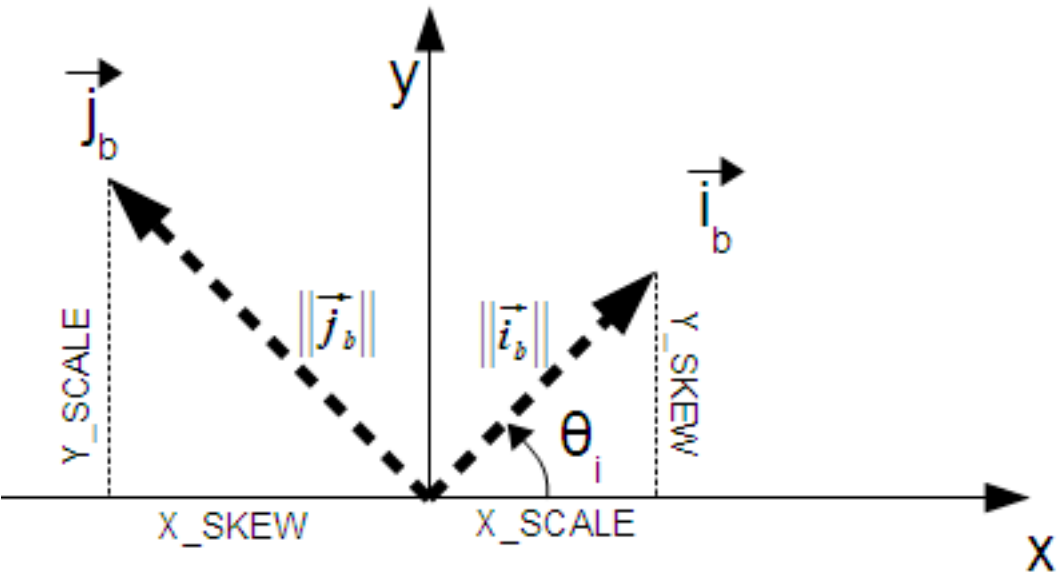
**Synopsis**

double precision **ST\_PixelWidth**(raster rast);

**Example:**

Example: Returns the pixel width of a raster. The pixel width is the width of the raster in units. The pixel width is the width of the raster in units. The pixel width is the width of the raster in units.





繁體中文:  $i$  繁體中文  
繁體中文:  $j$  繁體中文

繁體中文: 繁體中文

```
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     |

繁體中文: 0 繁體中文

```
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   |

繁體中文

ST\_PixelHeight, ST\_ScaleX, ST\_ScaleY, ST\_SkewX, ST\_SkewY

### 11.4.9 ST\_ScaleX

ST\_ScaleX — Scales the X dimension of a geometry.

#### Synopsis

float8 **ST\_ScaleX**(raster rast);

Geometry

Scales the X dimension of a geometry. The result is a geometry of the same type as the input. The result is a geometry of the same type as the input. The result is a geometry of the same type as the input.

Geometry

```
SELECT rid, ST_ScaleX(rast) As rastpixwidth
FROM dummy_rast;
```

| rid | rastpixwidth |
|-----|--------------|
| 1   | 2            |
| 2   | 0.05         |

Geometry

#### ST\_Width

### 11.4.10 ST\_ScaleY

ST\_ScaleY — Scales the Y dimension of a geometry.

#### Synopsis

float8 **ST\_ScaleY**(raster rast);

Geometry

Scales the Y dimension of a geometry. The result is a geometry of the same type as the input. The result is a geometry of the same type as the input. The result is a geometry of the same type as the input.

Geometry

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;
```

| rid | rastpixheight |
|-----|---------------|
| 1   | 3             |
| 2   | -0.05         |

图

ST\_Height

11.4.11 ST\_RasterToWorldCoord

ST\_RasterToWorldCoord — 返回栅格中指定像素的地理坐标 X, Y(经度, 纬度) 值。像素索引从 1 开始。

Synopsis

record **ST\_RasterToWorldCoord**(raster rast, integer xcolumn, integer yrow);

图

返回栅格中指定像素的地理坐标 X, Y(经度, 纬度) 值。像素索引从 1 开始。如果栅格是倾斜的，则返回的坐标将是倾斜后的坐标。如果栅格是未倾斜的，则返回的坐标将是未倾斜后的坐标。

2.1.0 版本引入。

图

```
-- 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 |

文档

[ST\\_RasterToWorldCoordX](#), [ST\\_RasterToWorldCoordY](#), [ST\\_SetSkew](#)

### 11.4.12 ST\_RasterToWorldCoordX

`ST_RasterToWorldCoordX` — 返回栅格像素的 X 世界坐标。像素索引从 1 开始。

#### Synopsis

```
float8 ST_RasterToWorldCoordX(raster rast, integer xcolumn);
float8 ST_RasterToWorldCoordX(raster rast, integer xcolumn, integer yrow);
```

文档

返回栅格像素的 X 世界坐标。像素索引从 1 开始。如果提供了 Y 行索引，则返回 Y 行索引对应的 X 世界坐标。如果未提供 Y 行索引，则返回 X 世界坐标的平均值。



#### Note

如果提供了 Y 行索引，则返回 Y 行索引对应的 X 世界坐标。如果未提供 Y 行索引，则返回 X 世界坐标的平均值。使用 `ST_ScaleX`, `ST_SkewX`, 和 `ST_SetSkew` 函数可以设置 X 世界坐标。

更新: 2.1.0 版本中 `ST_Raster2WorldCoordX` 函数被弃用。

文档

```
-- non-skewed raster providing column is sufficient
SELECT rid, ST_RasterToWorldCoordX(rast,1) As xlcoord,
       ST_RasterToWorldCoordX(rast,2) As x2coord,
       ST_ScaleX(rast) As pixelx
FROM dummy_rast;
```

| rid | xlcoord    | 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 xlcoord,
       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 | xlcoord    | x2coord   | pixelx |
|-----|------------|-----------|--------|
| 1   | 0.5        | 203.5     | 2      |
| 2   | 3427927.75 | 3428128.8 | 0.05   |

文档

[ST\\_ScaleX](#), [ST\\_RasterToWorldCoordY](#), [ST\\_SetSkew](#), [ST\\_SkewX](#)

### 11.4.13 ST\_RasterToWorldCoordY

`ST_RasterToWorldCoordY` — 返回栅格 `rast` 中 `yrow` 行对应的世界坐标 Y 值。返回类型为 `float8`。如果 `yrow` 为 1，则返回栅格的顶部 Y 值。

#### Synopsis

```
float8 ST_RasterToWorldCoordY(raster rast, integer yrow);
float8 ST_RasterToWorldCoordY(raster rast, integer xcolumn, integer yrow);
```

文档

`ST_RasterToWorldCoordY` 返回栅格 `rast` 中 `yrow` 行对应的世界坐标 Y 值。返回类型为 `float8`。如果 `yrow` 为 1，则返回栅格的顶部 Y 值。如果 `xcolumn` 和 `yrow` 都指定，则返回栅格中指定位置的像素值。



#### Note

如果栅格是倾斜的，Y 值将不是整数。在这种情况下，`ST_ScaleY`, `ST_SkewY`, `ST_SkewX` 函数将返回非整数 Y 值。如果栅格是倾斜的，Y 值将不是整数。

注意: 2.1.0 版本中 `ST_Raster2WorldCoordY` 函数已弃用。

文档

```
-- 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  |

[ST\\_ScaleY](#), [ST\\_RasterToWorldCoordX](#), [ST\\_SetSkew](#), [ST\\_SkewY](#)

### 11.4.14 ST\_Rotation

`ST_Rotation` — 返回栅格的旋转角度。

#### Synopsis

`float8 ST_Rotation(raster rast);`

返回栅格的旋转角度。如果栅格是正交的，则返回 0。如果栅格是 45 度旋转的，则返回 0.785398163397448。NaN 表示未知。如果栅格是 45 度旋转的，则返回 0.785398163397448。

```
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 |

[ST\\_SetRotation](#), [ST\\_SetScale](#), [ST\\_SetSkew](#)

### 11.4.15 ST\_SkewX

`ST_SkewX` — 返回栅格的 X 轴偏斜 (skew) 角度。

#### Synopsis

`float8 ST_SkewX(raster rast);`

返回栅格的 X 轴偏斜 (skew) 角度。如果栅格是正交的，则返回 0。如果栅格是 45 度偏斜的，则返回 0.785398163397448。

```
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 |

**ST\_GeoReference, ST\_SkewY, ST\_SetSkew**

11.4.16 ST\_SkewY

ST\_SkewY — Returns Y skew (in degrees) of a raster.

Synopsis

```
float8 ST_SkewY(raster rast);
```

Returns Y skew (in degrees) of a raster. Returns **NaN** if the raster is not skewed.

```
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       |
|     |       |       | : 0.0500000000       |
|     |       |       | : 0.0000000000       |
|     |       |       | : 0.0000000000       |
|     |       |       | : -0.0500000000      |
|     |       |       | : 3427927.7500000000 |
|     |       |       | : 5793244.0000000000 |

```

2 |      0 |      0 | 0.0500000000
   :      :      : 0.0000000000
   :      :      : 0.0000000000
   :      :      : -0.0500000000
   :      :      : 3427927.7500000000
   :      :      : 5793244.0000000000

```

☐☐

[ST\\_GeoReference](#), [ST\\_SkewX](#), [ST\\_SetSkew](#)

### 11.4.17 ST\_SRID

**ST\_SRID** — spatial\_ref\_sys 表, 返回 SRID 值。

#### Synopsis

integer **ST\_SRID**(raster rast);

☐☐

spatial\_ref\_sys 表, 返回 SRID 值。



#### Note

PostGIS 2.0 之前, 返回 SRID 值 -1 表示 0 值。

☐☐

```

SELECT ST_SRID(rast) As srid
FROM dummy_rast WHERE rid=1;

```

```

srid
-----
0

```

☐☐

Section [4.5](#), [ST\\_SRID](#)

### 11.4.18 ST\_Summary

**ST\_Summary** — 返回摘要信息。

#### Synopsis

text **ST\_Summary**(raster rast);



国国

国国国国国国国国国国国国国国国国.

2.1.0 国国国国国国国国国国国国.

国国

```

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)

```

国国

**ST\_MetaData, ST\_BandMetaData, ST\_Summary ST\_Extent**

### 11.4.19 ST\_UpperLeftX

ST\_UpperLeftX — 国国国国国国 X 国国国国国国国国国国国国国国国国.

#### Synopsis

float8 **ST\_UpperLeftX**(raster rast);

国国

国国国国国国 X 国国国国国国国国国国国国国国国国.

国国

```

SELECT rid, ST_UpperLeftX(rast) As ulx
FROM dummy_rast;

```

```

rid |      ulx
-----+-----
  1 |      0.5
  2 | 3427927.75

```

返回

[ST\\_UpperLeftY](#), [ST\\_GeoReference](#), [Box3D](#)

### 11.4.20 ST\_UpperLeftY

`ST_UpperLeftY` — 返回栅格的 Y 坐标值。

#### Synopsis

`float8 ST_UpperLeftY(raster rast);`

返回

返回栅格的 Y 坐标值。

返回

```
SELECT rid, ST_UpperLeftY(rast) As uly
FROM dummy_rast;
```

| rid | uly     |
|-----|---------|
| 1   | 0.5     |
| 2   | 5793244 |

返回

[ST\\_UpperLeftX](#), [ST\\_GeoReference](#), [Box3D](#)

### 11.4.21 ST\_Width

`ST_Width` — 返回栅格的宽度。

#### Synopsis

`integer ST_Width(raster rast);`

返回

返回栅格的宽度。

```
SELECT ST_Width(rast) As rastwidth
FROM dummy_rast WHERE rid=1;

rastwidth
-----
10
```

ST\_Height

11.4.22 ST\_WorldToRasterCoord

ST\_WorldToRasterCoord — X, Y(, )   
.

Synopsis

record **ST\_WorldToRasterCoord**(raster rast, geometry pt);  
record **ST\_WorldToRasterCoord**(raster rast, double precision longitude, double precision latitude);

X, Y(, )   
.

2.1.0 .

```
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
```

ST\_WorldToRasterCoordX, ST\_WorldToRasterCoordY, ST\_RasterToWorldCoordX, ST\_RasterToWorldCoordY  
ST\_SRID

### 11.4.23 ST\_WorldToRasterCoordX

ST\_WorldToRasterCoordX —  $\text{XXXXXX}$  (pt)  $\text{XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}$   
X, Y  $\text{XXXXXX}$  (xw, yw)  $\text{XXXXXXXXXX}$ .

## 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);
```



`x` (`pt`) `X`, `y` (`xw`, `yw`) . `(xw`, `yw`)  
`xw` `yw` . `xw` .

ST World2RasterCoordX



```
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        |



ST RasterToWorldCoordX, ST RasterToWorldCoordY, ST SRID

#### 11.4.24 ST\_WorldToRasterCoordY

ST WorldToRasterCoordY —  $\text{XXXXXX}$  (pt)  $\text{XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}$   
X, Y  $\text{XXXXXX}$  (xw, yw)  $\text{XXXXXXXXXX}$ .

## 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);
```

`ST_WorldToRasterCoordY` (pt) returns the Y coordinate of the point (xw, yw) in the raster. The raster is identified by the raster id (xw, yw) in the raster. The raster is identified by the raster id (xw, yw) in the raster.

`ST_WorldToRasterCoordY` returns the Y coordinate of the point (xw, yw) in the raster.

```
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 |

`ST_RasterToWorldCoordX`, `ST_RasterToWorldCoordY`, `ST_SRID`

## 11.5

### 11.5.1 ST\_BandMetaData

`ST_BandMetaData` — Returns basic meta data about a raster band. Columns returned: pixeltype, nodatavalue, isoutdb, path, outdbbandnum, filesize, filetimestamp.

#### Synopsis

- (1) record `ST_BandMetaData`(raster rast, integer band=1);
- (2) record `ST_BandMetaData`(raster rast, integer[] band);

Returns basic meta data about a raster band. Columns returned: pixeltype, nodatavalue, isoutdb, path, outdbbandnum, filesize, filetimestamp.



**Note**  
If band has no NODATA value, nodatavalue are NULL.



**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.

Example 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    |              |

Example 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 | outdbbandnum | filesize | filetimestamp | path                                                                      |
|---------|-----------|-------------|---------|--------------|----------|---------------|---------------------------------------------------------------------------|
| 1       | 8BUI      |             | t       |              |          |               | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif |
| 3       | 8BUI      |             | t       |              |          |               | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif |
| 2       | 8BUI      |             | t       |              |          |               | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif |



**ST\_MetaData, ST\_BandPixelType**

## 11.5.2 ST\_BandNoDataValue

`ST_BandNoDataValue` — 返回指定 NODATA 值的栅格带的值。返回值的范围是 1 到 255。

### Synopsis

double precision **ST\_BandNoDataValue**(raster rast, integer bandnum=1);

返回

返回 NODATA 值的栅格带的值。

返回

```
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      |

返回

**ST\_NumBands**

## 11.5.3 ST\_BandIsNoData

`ST_BandIsNoData` — 返回指定 NODATA 值的栅格带的值。

### Synopsis

boolean **ST\_BandIsNoData**(raster rast, integer band, boolean forceChecking=true);

boolean **ST\_BandIsNoData**(raster rast, boolean forceChecking=true);

返回

返回 NODATA 值的栅格带的值。返回值的范围是 1 到 255。如果返回 TRUE，则表示该值在 NODATA 范围内。如果返回 FALSE，则表示该值不在 NODATA 范围内。

2.0.0 版本引入。

### Note



在 2.0.0 版本中，`ST_BandIsNoData` 函数默认返回 TRUE。如果希望返回 FALSE，可以使用 `ST_SetBandIsNoData` 函数。例如，`ST_SetBandIsNoData(rast, 1, TRUE)` 将第一个带的值设置为 NODATA。使用 `ST_SetBandIsNoData` 函数时，需要指定要设置的带编号和要设置的值（TRUE 或 FALSE）。

❏❏

```
-- 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
```

❏❏

**ST\_BandNoDataValue, ST\_NumBands, ST\_SetBandNoDataValue, ST\_SetBandIsNoData**



### 11.5.4 ST\_BandPath

`ST_BandPath` — Returns the path to the band file. `bandnum` defaults to 1.

#### Synopsis

text **ST\_BandPath**(raster rast, integer bandnum=1);

`ST_BandPath`

DB `ST_BandPath`.

`ST_BandPath`

`ST_BandPath`

### 11.5.5 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);

`ST_BandFileSize`

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

`ST_BandFileSize`

```
SELECT ST_BandFileSize(rast,1) FROM dummy_rast WHERE rid = 1;
```

```
st_bandfilesize
-----
240574
```

### 11.5.6 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);

返回

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

示例

```
SELECT ST_BandFileTimestamp(rast,1) FROM dummy_rast WHERE rid = 1;
```

```
st_bandfiletimestamp
-----
1521807257
```

### 11.5.7 ST\_BandPixelType

**ST\_BandPixelType** — 返回指定带的像素数据类型。bandnum 指定带的编号，默认为 1。

#### Synopsis

text **ST\_BandPixelType**(raster rast, integer bandnum=1);

返回

Returns name describing data type and size of values stored in each cell of given band.

11 返回的字符串格式为：`bandnum 数据类型 大小`。

- 1BB - 1 字节
- 2BUI - 2 字节无符号整数
- 4BUI - 4 字节无符号整数
- 8BSI - 8 字节有符号整数
- 8BUI - 8 字节无符号整数
- 16BSI - 16 字节有符号整数
- 16BUI - 16 字节无符号整数

- 32BSI - 32 signed integers
- 32BUI - 32 unsigned integers
- 32BF - 32 IEEE 754-2008 single-precision floating point numbers
- 64BF - 64 IEEE 754-2008 double-precision floating point numbers

Example

```
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   |

Example

**ST\_NumBands**

**11.5.8 ST\_MinPossibleValue**

ST\_MinPossibleValue — Returns the minimum possible value for a pixel type.

**Synopsis**

```
integer ST_MinPossibleValue(text pixeltype);
```

Example

Return the minimum possible value for a 16-bit signed integer.

Example

```
SELECT ST_MinPossibleValue('16BSI');

 st_minpossiblevalue
-----
-32768

SELECT ST_MinPossibleValue('8BUI');

 st_minpossiblevalue
-----
0
```



例

```
-- 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, .. |

例

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroid](#), [ST\\_PixelAsCentroids](#), [ST\\_Intersection](#), [ST\\_AsText](#)

## 11.6.2 ST\_PixelAsPolygons

`ST_PixelAsPolygons` — 将栅格像素转换为多边形。返回 X, Y 坐标的多边形。

### Synopsis

setof record **ST\_PixelAsPolygons**(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);

例

返回记录格式: *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` 返回的多边形是 `ST_DumpAsPolygons` 返回的多边形的子集。

2.0.0 版本引入。

版本: 2.1.0 版本引入 `exclude_nodata_value` 参数。

版本: 2.1.1 版本引入 `exclude_nodata_value` 参数。

```
-- get raster pixel polygon
SELECT (gv).x, (gv).y, (gv).val, ST_AsText((gv).geom) geom
FROM (SELECT ST_PixelAsPolygons(
    ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.001,
    -0.001, 0.001, 0.001, 4269),
    '8BUI'::text, 1, 0),
    2, 2, 10),
    1, 1, NULL)
) gv
) foo;
```

| x | y | val | geom                                                                   |
|---|---|-----|------------------------------------------------------------------------|
| 1 | 1 |     | POLYGON((0 0,0.001 0.001,0.002 0,0.001 -0.001,0 0))                    |
| 1 | 2 | 1   | POLYGON((0.001 -0.001,0.002 0,0.003 -0.001,0.002 -0.002,0.001 -0.001)) |
| 2 | 1 | 1   | POLYGON((0.001 0.001,0.002 0.002,0.003 0.001,0.002 0,0.001 0.001))     |
| 2 | 2 | 10  | POLYGON((0.002 0,0.003 0.001,0.004 0,0.003 -0.001,0.002 0))            |

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroid](#), [ST\\_PixelAsCentroids](#), [ST\\_AsText](#)

11.6.3 ST\_PixelAsPoint

ST\_PixelAsPoint — Get the geometry of a pixel from a raster.

Synopsis

geometry **ST\_PixelAsPoint**(raster rast, integer columnx, integer rowy);

Get the geometry of a pixel from a raster.

2.1.0 Get the geometry of a pixel from a raster.

```
SELECT ST_AsText(ST_PixelAsPoint(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;
```

| st_astext      |
|----------------|
| POINT(0.5 0.5) |

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroid](#), [ST\\_PixelAsCentroids](#)

### 11.6.4 ST\_PixelAsPoints

`ST_PixelAsPoints` — Returns the `x`, `y` coordinates and the value of the pixel at the given location.

#### Synopsis

setof record `ST_PixelAsPoints`(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);

Returns a set of records with columns `x`, `y` and `val`. The `val` column is of type `double precision`.

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.

2.1.0

2.1.1 `exclude_nodata_value`

```
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)  |

函数

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsCentroid](#), [ST\\_PixelAsCentroids](#)

### 11.6.5 ST\_PixelAsCentroid

`ST_PixelAsCentroid` — 返回栅格像素 (x, y) 的质心。

#### Synopsis

geometry **ST\_PixelAsCentroid**(raster rast, integer x, integer y);

函数

返回栅格像素 (x, y) 的质心。

版本: 2.1.0 引入 C 函数。

2.1.0 引入。

函数

```
SELECT ST_AsText(ST_PixelAsCentroid(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;
```

```
st_astext
-----
POINT(1.5 2)
```

函数

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroids](#)

### 11.6.6 ST\_PixelAsCentroids

`ST_PixelAsCentroids` — 返回栅格像素 (x, y) 的质心 X, Y 坐标。

#### Synopsis

setof record **ST\_PixelAsCentroids**(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);



(X, Y ).

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.

- 2.1.0 C
- 2.1.1 *exclude\_nodata\_value*
- 2.1.0

```
--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)
```

*ST\_DumpAsPolygons*, *ST\_PixelAsPolygon*, *ST\_PixelAsPolygons*, *ST\_PixelAsPoint*, *ST\_PixelAsPoints*, *ST\_Pixel*

### 11.6.7 ST\_Value

**ST\_Value** — 返回 raster 在 geometry 点处的值。如果 geometry 点位于 raster 的 nodata 区域，则返回 exclude\_nodata\_value 的值。如果 exclude\_nodata\_value 为 true，则返回 nodata 的值。如果 exclude\_nodata\_value 为 false，则返回 raster 的值。如果 geometry 点位于 raster 的边界上，则返回 raster 的值。如果 geometry 点位于 raster 的边界上，则返回 raster 的值。

#### 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);
```

返回

返回 raster 在 geometry 点处的值。如果 geometry 点位于 raster 的 nodata 区域，则返回 exclude\_nodata\_value 的值。如果 exclude\_nodata\_value 为 true，则返回 nodata 的值。如果 exclude\_nodata\_value 为 false，则返回 raster 的值。如果 geometry 点位于 raster 的边界上，则返回 raster 的值。如果 geometry 点位于 raster 的边界上，则返回 raster 的值。

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.

2.1.0 版本: 返回 raster 在 geometry 点处的值。如果 geometry 点位于 raster 的 nodata 区域，则返回 exclude\_nodata\_value 的值。如果 exclude\_nodata\_value 为 true，则返回 nodata 的值。如果 exclude\_nodata\_value 为 false，则返回 raster 的值。如果 geometry 点位于 raster 的边界上，则返回 raster 的值。如果 geometry 点位于 raster 的边界上，则返回 raster 的值。

返回

```
-- get raster values at particular postgis geometry points
-- the srid of your geometry should be same as for your raster
SELECT rid, ST_Value(rast, foo.pt_geom) As b1pval, ST_Value(rast, 2, foo.pt_geom) As b2pval
FROM dummy_rast CROSS JOIN (SELECT ST_SetSRID(ST_Point(3427927.77, 5793243.76), 0) As
pt_geom) As foo
WHERE rid=2;
```

```
rid | b1pval | b2pval
-----+-----+-----
2 | 252 | 79
```

```
-- general fictitious example using a real table
SELECT rid, ST_Value(rast, 3, sometable.geom) As b3pval
FROM sometable
WHERE ST_Intersects(rast,sometable.geom);
```

```
SELECT rid, ST_Value(rast, 1, 1, 1) As b1pval,
ST_Value(rast, 2, 1, 1) As b2pval, ST_Value(rast, 3, 1, 1) As b3pval
FROM dummy_rast
WHERE rid=2;
```

```
rid | b1pval | b2pval | b3pval
-----+-----+-----+-----
2 | 253 | 78 | 70
```

```

--- Get all values in bands 1,2,3 of each pixel --
SELECT x, y, ST_Value(rast, 1, x, y) As b1val,
       ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1, 1000) As x CROSS JOIN generate_series(1, 1000) As y
WHERE rid = 2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);

```

| x | y | b1val | b2val | b3val |
|---|---|-------|-------|-------|
| 1 | 1 | 253   | 78    | 70    |
| 1 | 2 | 253   | 96    | 80    |
| 1 | 3 | 250   | 99    | 90    |
| 1 | 4 | 251   | 89    | 77    |
| 1 | 5 | 252   | 79    | 62    |
| 2 | 1 | 254   | 98    | 86    |
| 2 | 2 | 254   | 118   | 108   |
| : |   |       |       |       |
| : |   |       |       |       |

```

--- Get all values in bands 1,2,3 of each pixel same as above but returning the upper left ←
point point of each pixel --
SELECT ST_AsText(ST_SetSRID(
  ST_Point(ST_UpperLeftX(rast) + ST_ScaleX(rast)*x,
    ST_UpperLeftY(rast) + ST_ScaleY(rast)*y),
    ST_SRID(rast))) As uplpt
  , ST_Value(rast, 1, x, y) As b1val,
  ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid = 2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);

```

| uplpt                       | b1val | b2val | b3val |
|-----------------------------|-------|-------|-------|
| POINT(3427929.25 5793245.5) | 253   | 78    | 70    |
| POINT(3427929.25 5793247)   | 253   | 96    | 80    |
| POINT(3427929.25 5793248.5) | 250   | 99    | 90    |
| :                           |       |       |       |

```

--- Get a polygon formed by union of all pixels
that fall in a particular value range and intersect particular polygon --
SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
  ST_UpperLeftX(rast), ST_UpperLeftY(rast),
  ST_UpperLeftX(rast) + ST_ScaleX(rast),
  ST_UpperLeftY(rast) + ST_ScaleY(rast), 0
  ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
  ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid = 2
AND x <= ST_Width(rast) AND y <= ST_Height(rast)) As foo
WHERE
  ST_Intersects(
    pixpolyg,
    ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928 ←
  5793243.75,3427928 5793244))',0)

```

```
) AND b2val != 254;
```

```
shadow
```

```
-----
MULTIPOLYGON(((3427928 5793243.9,3427928 5793243.85,3427927.95 5793243.85,3427927.95  ←
5793243.9,
3427927.95 5793243.95,3427928 5793243.95,3427928.05 5793243.95,3427928.05  ←
5793243.9,3427928 5793243.9)),((3427927.95 5793243.9,3427927.95 579324
3.85,3427927.9 5793243.85,3427927.85 5793243.85,3427927.85 5793243.9,3427927.9  ←
5793243.9,3427927.9 5793243.95,
3427927.95 5793243.95,3427927.95 5793243.9)),((3427927.85 5793243.75,3427927.85  ←
5793243.7,3427927.8 5793243.7,3427927.8 5793243.75
,3427927.8 5793243.8,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85  ←
5793243.8,3427927.85 5793243.75)),
((3427928.05 5793243.75,3427928.05 5793243.7,3427928 5793243.7,3427927.95  ←
5793243.7,3427927.95 5793243.75,3427927.95 5793243.8,3427
927.95 5793243.85,3427928 5793243.85,3427928 5793243.8,3427928.05 5793243.8,
3427928.05 5793243.75)),((3427927.95 5793243.75,3427927.95 5793243.7,3427927.9  ←
5793243.7,3427927.85 5793243.7,
3427927.85 5793243.75,3427927.85 5793243.8,3427927.85 5793243.85,3427927.9 5793243.85,
3427927.95 5793243.85,3427927.95 5793243.8,3427927.95 5793243.75)))
```

```
--- Checking all the pixels of a large raster tile can take a long time.
--- You can dramatically improve speed at some lose of precision by orders of magnitude
-- by sampling pixels using the step optional parameter of generate_series.
-- This next example does the same as previous but by checking 1 for every 4 (2x2) pixels  ←
and putting in the last checked
-- putting in the checked pixel as the value for subsequent 4
```

```
SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
    ST_UpperLeftX(rast), ST_UpperLeftY(rast),
    ST_UpperLeftX(rast) + ST_ScaleX(rast)*2,
    ST_UpperLeftY(rast) + ST_ScaleY(rast)*2, 0
    ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
    ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
FROM dummy_rast CROSS JOIN
generate_series(1,1000,2) As x CROSS JOIN generate_series(1,1000,2) As y
WHERE rid = 2
AND x <= ST_Width(rast) AND y <= ST_Height(rast) ) As foo
WHERE
    ST_Intersects(
        pixpolyg,
        ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928  ←
5793243.75,3427928 5793244))',0)
    ) AND b2val != 254;
```

```
shadow
```

```
-----
MULTIPOLYGON(((3427927.9 5793243.85,3427927.8 5793243.85,3427927.8 5793243.95,
3427927.9 5793243.95,3427928 5793243.95,3427928.1 5793243.95,3427928.1 5793243.85,3427928  ←
5793243.85,3427927.9 5793243.85)),
((3427927.9 5793243.65,3427927.8 5793243.65,3427927.8 5793243.75,3427927.8  ←
5793243.85,3427927.9 5793243.85,
3427928 5793243.85,3427928 5793243.75,3427928.1 5793243.75,3427928.1 5793243.65,3427928  ←
5793243.65,3427927.9 5793243.65)))
```



ST\_DumpValues, 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\_Intersect,  
ST\_Intersection

### 11.6.8 ST\_NearestValue

[illegible]

## Synopsis

```
double precision ST_NearestValue(raster rast, integer bandnum, geometry pt, boolean exclude_nodata_val
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_valu
```



columnx, rowy , NODATA , columnx, rowy , NODATA , columnx, rowy , NODATA .

```

1 1, bandnum 1. exclude_nodata_value
nodata. exclude_nodata_value
, .

```

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.



### Note

ST NearestValue  ST Value .

[illegible]

```

),
  2, 3, 0.
),
  3, 5, 0.
),
  4, 2, 0.
),
  5, 4, 0.
) AS rast
) AS foo

value | nearestvalue
-----+-----
1 | 1
```

```
-- pixel 2x3 is NODATA
SELECT
  ST_Value(rast, 2, 3) AS value,
  ST_NearestValue(rast, 2, 3) AS nearestvalue
FROM (
  SELECT
    ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_SetValue(
            ST_SetValue(
              ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
              ),
              1, 1, 0.
            ),
            2, 3, 0.
          ),
          3, 5, 0.
        ),
        4, 2, 0.
      ),
      5, 4, 0.
    ) AS rast
  ) AS foo

value | nearestvalue
-----+-----
| 1
```



ST\_Neighborhood, ST\_Value

11.6.9 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);

⌘⌘

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

⌘⌘

```
--
-- 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 => 'l6BSI',
    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)
```

⌘⌘

**ST\_Value, ST\_SetSRID**

### 11.6.10 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);

⌘⌘

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

⌘⌘

```
--
-- 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 => 'l6BSI',
    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)
```





## ST\_Value, ST\_SetSRID

### 11.6.11 ST\_Neighborhood

ST\_Neighborhood — columnx □ rowy, □□□□□□□□□□□□□□□□□□□□□□□□□□□□  
 □□□□□□□□□□□□ NODATA □□□□□□□□□□□□□□□□ 2 □□□□□□□□□□.

## 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);
```

[illegible]

```

bandnum 1 exclude_nodata_value 1
nodata exclude_nodata_value

```



### Note

$\text{distanceX} \times 2 + \text{distanceY} + 1$



### Note

ST\_Min4ma, ST\_Sum4ma, ST\_Mean4ma  $\times \times \times \times \times \times \times \times \times \times \times \times \times \times \times$  2  $\times \times \times \times \times \times \times \times \times$   
 $\times \times \times \times \times \times \times \times \times \times$ .

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.



```
-- 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}}
```

[ST\\_NearestValue](#), [ST\\_Min4ma](#), [ST\\_Max4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Range4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

### 11.6.12 ST\_SetValue

**ST\_SetValue** — Set the value of a pixel in a raster. The pixel is identified by its column and row. The value is set to the newvalue. The pixel is identified by its column and row. The value is set to the newvalue.

#### 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);

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.

**ST\_SetValue**: 2.1.0 **ST\_SetValue**(raster rast, integer bandnum, geometry geom, double precision newvalue);  
**ST\_SetValues**: 2.1.0 **ST\_SetValues**(raster rast, geometry geom, double precision newvalue);

```
-- 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 ;
```

[ST\\_Value](#), [ST\\_DumpAsPolygons](#)

### 11.6.13 ST\_SetValues

ST\_SetValues — Set values in a raster.

#### 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);

**Parameters**  
**newvalueset** (array) An array of values to set in the raster.

**keepnodata** boolean, NODATA If TRUE, NODATA values in newvalueset are preserved in the output raster.

**1** If columnx, rowy are specified, newvalueset is a 1D array of values. If newvalueset is a 2D array, noset is a 2D array of booleans (PostgreSQL NULL values are treated as FALSE). If 1 is specified.

**2** If 1 is specified, noset is a 1D array of booleans. If nosetvalue is specified, newvalueset is a 1D array of values. If 2 is specified.

**3** If columnx, rowy, width, height are specified, newvalue is a single value. If 3 is specified.

**4** If rast is a raster, newvalue is a single value. If 4 is specified.

**5** If rast is a raster, geomval is a 1D array of POINT or MULTIPOINT values. If 5 is specified.

2.1.0 Added.

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 | 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 | 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   |

```

/*
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[3][3], -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 --
            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   |

Example 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   |

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, ←<br>NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL}}") |
| 1   | 2   | (1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{NULL, ←<br>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, ←<br>NULL,NULL},{NULL,NULL,NULL,NULL,NULL}}")                      |
| 1   | 4   | (1,"{{4,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL, ←<br>NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,4}}")    |

(4 rows)

geomvals geomvals 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},{ ←<br>NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}") |

(1 row)

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 = 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},{ ↵<br>NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}") |

(1 row)

**ST\_Value, ST\_SetValue, ST\_PixelAsPolygons**

### 11.6.14 ST\_DumpValues

ST\_DumpValues — 2

#### 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 );

2 (, ). nband NULL  
,.

2.1.0.

```
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)

**ST\_Value, ST\_SetValue, ST\_SetValues**

**11.6.15 ST\_PixelOfValue**

ST\_PixelOfValue — columnx, rowy.

**Synopsis**

```
setof record ST_PixelOfValue( raster rast , integer nband , double precision[] search , boolean ex-  
clude_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 ex-  
clude_nodata_value=true );  
setof record ST_PixelOfValue( raster rast , double precision search , boolean exclude_nodata_value=true  
);
```

columnx, rowy. 1

2.1.0.

```
SELECT
  (pixels).*
FROM (
  SELECT
    ST_PixelOfValue(
      ST_SetValue(
        ST_SetValue(
          ST_SetValue(
            ST_SetValue(
              ST_SetValue(
                ST_AddBand(
                  ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                  '8BUI'::text, 1, 0
                ),
                1, 1, 0
              ),
              2, 3, 0
            ),
            3, 5, 0
          ),
          4, 2, 0
        ),
        5, 4, 255
      ),
      1, ARRAY[1, 255]) AS pixels
) AS foo
```

| val | x | y |
|-----|---|---|
| 1   | 1 | 2 |
| 1   | 1 | 3 |
| 1   | 1 | 4 |
| 1   | 1 | 5 |
| 1   | 2 | 1 |
| 1   | 2 | 2 |
| 1   | 2 | 4 |
| 1   | 2 | 5 |
| 1   | 3 | 1 |
| 1   | 3 | 2 |
| 1   | 3 | 3 |
| 1   | 3 | 4 |
| 1   | 4 | 1 |
| 1   | 4 | 3 |
| 1   | 4 | 4 |
| 1   | 4 | 5 |
| 1   | 5 | 1 |
| 1   | 5 | 2 |
| 1   | 5 | 3 |
| 255 | 5 | 4 |
| 1   | 5 | 5 |

## 11.7 地理参考

### 11.7.1 ST\_SetGeoReference

`ST_SetGeoReference` — 将地理参考信息设置到 6 个参数。支持 GDAL 和 ESRI 两种格式。GDAL 格式。

#### 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`);

参数

`georefcoords` 6 个参数。'GDAL' 或 'ESRI' 格式。GDAL 格式。6 个参数。NULL 表示未设置。

GDAL 格式:

GDAL:

```
scalex skewy skewx scaley upperleftx upperlefty
```

ESRI:

```
scalex skewy skewx scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5
```



#### Note

在 PostgreSQL 12 中，`ST_SetGeoReference` 函数在 2.1.0 版本中被弃用。建议使用 `ST_SetSRID` 函数。

更新: 2.1.0 版本中 `ST_SetGeoReference(raster, double precision, ...)` 函数被弃用。

示例

```
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\_Metadata(ST\_SetGeoReference(rast, 1, 1, 10, -10, 0.001, 0.001))).\*

FROM foo

| rid | upperleftx<br>skewy | srid                | numbands | upperlefty | width | height | scalex | scaley | skewx |   |
|-----|---------------------|---------------------|----------|------------|-------|--------|--------|--------|-------|---|
| 0   | 0                   | 0                   | 0        | 0          | 5     | 5      | 1      | -1     | 0     | ↔ |
| 1   | 0                   | 0                   | 0.1      | 0.1        | 5     | 5      | 10     | -10    | 0     | ↔ |
| 2   | 0.09999999999999996 | 0.09999999999999996 |          |            | 5     | 5      | 10     | -10    | 0     | ↔ |
| 3   | 0.001               | 0                   | 1        | 1          | 5     | 5      | 10     | -10    | 0.001 | ↔ |

ST\_GeoReference, ST\_ScaleX, ST\_ScaleY, ST\_UpperLeftX, ST\_UpperLeftY

11.7.2 ST\_SetRotation

ST\_SetRotation —

Synopsis

raster **ST\_SetRotation**(raster rast, float8 rotation);

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                  |   |

ST\_Rotation, ST\_ScaleX, ST\_ScaleY, ST\_SkewX, ST\_SkewY



### 11.7.3 ST\_SetScale

ST\_SetScale — X 與 Y 座標縮放倍率。 縮放/縮減/放大座標。

#### Synopsis

```
raster ST_SetScale(raster rast, float8 xy);
raster ST_SetScale(raster rast, float8 x, float8 y);
```

說明

X 與 Y 座標縮放倍率。 縮放/縮減/放大座標。 縮放座標, X 與 Y 座標縮放倍率。



#### Note

ST\_SetScale 與 ST\_Rescale 類似。 ST\_Rescale 與 ST\_SetScale 不同。 ST\_Rescale (縮放) 與 ST\_SetScale 不同。 ST\_Rescale 與 ST\_SetScale 不同。 ST\_SetScale 與 ST\_Rescale 不同。

說明: 2.0.0 版 WKTRaster 與 ST\_SetPixelSize 類似。 2.0.0 版 ST\_SetPixelSize 與 ST\_SetScale 類似。

說明

```
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) |

說明

ST\_ScaleX, ST\_ScaleY, Box3D

### 11.7.4 ST\_SetSkew

`ST_SetSkew` — Set the skew of a 2D geometry. The skew is defined as the angle between the X and Y axes. The skew is measured in degrees. The skew is set to the value of the `skew` parameter. The skew is set to the value of the `skew` parameter. The skew is set to the value of the `skew` parameter.

#### Synopsis

```
raster ST_SetSkew(raster rast, float8 skewxy);
raster ST_SetSkew(raster rast, float8 skewx, float8 skewy);
```

Set the skew of a 2D geometry. The skew is defined as the angle between the X and Y axes. The skew is measured in degrees. The skew is set to the value of the `skew` parameter. The skew is set to the value of the `skew` parameter. The skew is set to the value of the `skew` parameter.

```
-- Example 1
UPDATE dummy_rast SET rast = ST_SetSkew(rast,1,2) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

| rid | skewx | skewy | georef                                                                                                 |
|-----|-------|-------|--------------------------------------------------------------------------------------------------------|
| 1   | 1     | 2     | 2.0000000000<br>: 2.0000000000<br>: 1.0000000000<br>: 3.0000000000<br>: 0.5000000000<br>: 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<br>: 0.0000000000<br>: 0.0000000000<br>: 3.0000000000<br>: 0.5000000000<br>: 0.5000000000 |

[ST\\_GeoReference](#), [ST\\_SetGeoReference](#), [ST\\_SkewX](#), [ST\\_SkewY](#)

### 11.7.5 ST\_SetSRID

`ST_SetSRID` — Set SRID of `spatial_ref_sys` SRID.

#### Synopsis

```
raster ST_SetSRID(raster rast, integer srid);
```

SRID.



#### Note

SRID. SRID.

Section 4.5, [ST\\_SRID](#)

### 11.7.6 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);
```

Set the value of the upper left corner of raster to the projected X and Y coordinates

```
SELECT ST_SetUpperLeft(rast, -71.01, 42.37)
FROM dummy_rast
WHERE rid = 2;
```

[ST\\_UpperLeftX](#), [ST\\_UpperLeftY](#)

### 11.7.7 ST\_Resample

**ST\_Resample** — Resample a raster to a new grid, width, height, gridx, gridy, scalex, scaley, skewx, skewy, text algorithm=NearestNeighbor, double precision maxerr=0.125).

#### 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 max-
err=0.125, boolean usescale=true);
raster ST_Resample(raster rast, raster ref, boolean usescale, text algorithm=NearestNeighbor, dou-
ble precision maxerr=0.125);
```

**Parameters**

**width**, **height** (width & height), **gridx** & **gridy**, **scalex**, **scaley**, **skewx** & **skewy** (scalex, scaley, skewx & skewy) are optional. **SRID** is optional.

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.

**maxerr** is optional. 0.125 is the default.



#### Note

GDAL Warp resampling methods

2.0.0 GDAL 1.6.1

Enhanced: 3.4.0 max and min resampling options added

❏❏

```
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       |

❏❏

[ST\\_Rescale](#), [ST\\_Resize](#), [ST\\_Transform](#)

### 11.7.8 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 scalex, 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);
```

❏❏

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.

`scalex` `scaley` `width`, `height`, `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

## GDAL Warp resampling methods



### Note

ST\_Rescale [REDACTED] ST\_SetScale [REDACTED].  
ST\_SetScale [REDACTED] ([REDACTED]) [REDACTED]  
[REDACTED]. ST\_Rescale [REDACTED]  
[REDACTED]. ST\_SetScale [REDACTED].

2.0.0 20200818. GDAL 1.6.1 20100501.

Enhanced: 3.4.0 max and min resampling options added

SRID: 2.1.0 SRID



0.001 0.0015

```
-- 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
```



ST Resize, ST Resample, ST SetScale, ST ScaleX, ST ScaleY, ST Transform

### 11.7.9 ST\_Reskew

ST\_Reskew — 指定した傾き (skew) を持つラスタを生成する。NearestNeighbor、Bilinear、Cubic、CubicSpline、Lanczos、NearestNeighbor のいずれかの再サンプリング方法を指定する。

#### 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);
```

戻り値

指定した傾き (skew) を持つラスタを生成する。NearestNeighbor、Bilinear、Cubic、CubicSpline、Lanczos、NearestNeighbor のいずれかの再サンプリング方法を指定する。

skewx は skewy を指定する。

maxerr は再サンプリングの最大エラー率を指定する。

maxerr は 0.125 を指定する。



#### Note

GDAL Warp resampling methods を参照する。



#### Note

ST\_Reskew は ST\_SetSkew を使用して傾きを設定する。ST\_SetSkew はラスタの傾きを指定する (傾き) を指定する。ST\_Reskew はラスタの傾きを指定する (傾き) を指定する。ST\_SetSkew はラスタの傾きを指定する (傾き) を指定する。

2.0.0 から 2.1.0 までの GDAL 1.6.1 を使用する場合。

2.1.0 から 2.2.0 までの SRID を指定する。

戻り値

傾き 0.0 から 0.0015 を指定する。

```
-- 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
```

繁體

[ST\\_Resample](#), [ST\\_Rescale](#), [ST\\_SetSkew](#), [ST\\_SetRotation](#), [ST\\_SkewX](#), [ST\\_SkewY](#), [ST\\_Transform](#)

### 11.7.10 ST\_SnapToGrid

`ST_SnapToGrid` — 將輸入的栅格重采样到指定的栅格。 `NearestNeighbor`(`栅格`), `Bilinear`, `Cubic`, `CubicSpline` 或 `Lanczos` 重采样方法。 `NearestNeighbor` 是默认方法。

#### 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 scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

繁體

`gridx` & `gridy` 是栅格的宽度和高度。 `scalex` & `scaley` 是栅格的宽度和高度的缩放比例。 `NearestNeighbor`(`栅格`), `Bilinear`, `Cubic`, `CubicSpline` 或 `Lanczos` 重采样方法。 `NearestNeighbor` 是默认方法。

`gridx` & `gridy` 是栅格的宽度和高度。 `scalex` & `scaley` 是栅格的宽度和高度的缩放比例。

You can optionally define the pixel size of the new grid with `scalex` and `scaley`.

`maxerr` 是栅格的精度。默认值是 0.125。

`maxerr` 是栅格的精度。默认值是 0.125。



#### Note

有关 [GDAL Warp resampling methods](#) 的更多信息。



#### Note

有关 [ST\\_Resample](#) 的更多信息。

2.0.0 版本中，`ST_SnapToGrid` 依赖于 `GDAL 1.6.1`。

注意：2.1.0 版本中，`SRID` 是必需的。



例

以下 SQL 语句展示了如何计算栅格的左上角 X 坐标。

```
-- 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
```

例

[ST\\_Resample](#), [ST\\_Rescale](#), [ST\\_UpperLeftX](#), [ST\\_UpperLeftY](#)

### 11.7.11 ST\_Resize

`ST_Resize` — 将栅格重新调整为指定的宽度和高度。

#### 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);

例

以下 SQL 语句展示了如何将栅格重新调整为指定的宽度和高度。

NearestNeighbor(最近邻), Bilinear, Cubic, CubicSpline 和 Lanczos 是支持的插值算法。NearestNeighbor 是默认值。

选项 1 指定了新的宽度和高度。

选项 2 指定了新的宽度和高度为原宽度和高度的百分比。0 到 1 之间的值表示百分比。

选项 3 指定了新的宽度和高度为原宽度和高度的百分比。默认值为 20%。

2.1.0 版本中，GDAL 1.6.1 版本引入了此函数。

SQL

```
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    |  |
| 2        | 0          | 0          | 500   | 100    | 1      | -1     | 0     | 0     | 0    |  |
| 3        | 0          | 0          | 250   | 900    | 1      | -1     | 0     | 0     | 0    |  |

(3 rows)

SQL

**ST\_Resample, ST\_Rescale, ST\_Reskew, ST\_SnapToGrid**

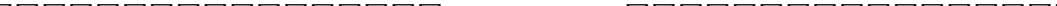
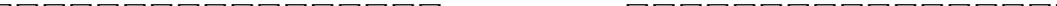
11.7.12 ST\_Transform

ST Transform — NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos 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);
```



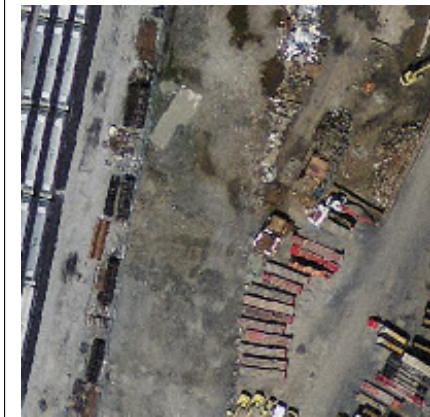

 (pixel warp) 
 NearestNeighbor, maxerror 0.125.

GDAL Warp resampling methods: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', 'Lanczos'.

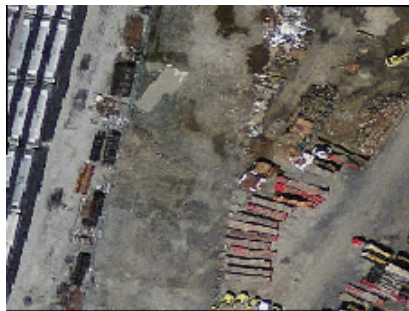
ST\_Transform(*geom*) ST\_SetSRID(*geom*, *srid*). ST\_Transform(*geom*, *srid*)  
 (*geom* is a geometry object, *srid* is an SRID value) *geom*, ST\_SetSRID(*geom*,  
*srid*) SRID *srid*.

XXXXXXXX, 3 alignto XXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXXXXXXX  
XX (SRID) XXXXXXXX, (ST SameAlignment = TRUE XXX) XXXXXXXXXXXXXXXX.

| w_before | w_after | h_before | h_after |
|----------|---------|----------|---------|
| 200      | 228     | 200      | 170     |



原始影像 (mass\_stm)



WGS84 影像 (wgs\_84) 影像



影像 NN 影像 Bilinear 影像  
影像 WGS84 影像  
(wgs\_84\_bilin) 影像

### 範例: 範例 3

使用 ST\_Transform(raster, srid) 與 ST\_Transform(raster, alignto) 對影像進行座標轉換。

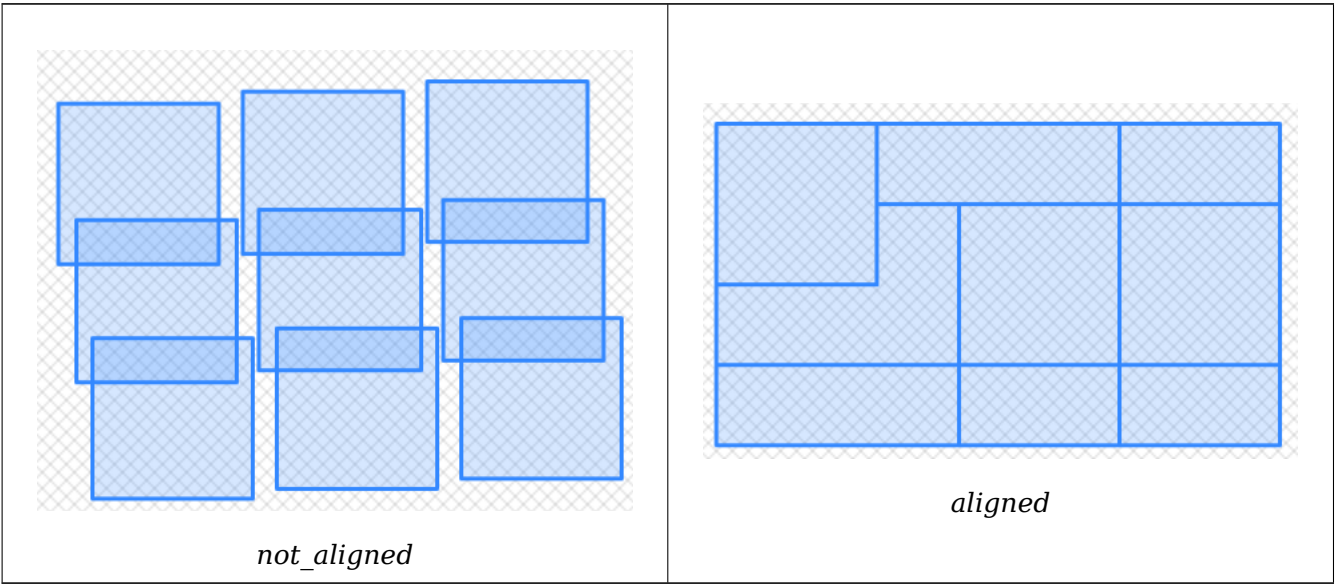
```
WITH foo AS (
  SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 600000, 100, -100, 0, 0,
    2163), 1, '16BUI', 1, 0) AS rast UNION ALL
  SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 600000, 100, -100, 0, 0, 2163),
    1, '16BUI', 2, 0) AS rast UNION ALL
  SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 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
```



**ST\_Transform, ST\_SetSRID**

11.8

11.8.1 ST\_SetBandNoDataValue

ST\_SetBandNoDataValue — NODATA.   
 1.   
 NODATA, 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);
```

文档

文档 NODATA 文档。 文档 1 文档。 文档 **ST\_Polygon**, **ST\_DumpAsPolygons**, 文档 **ST\_PixelAs...**() 文档。

文档

```
-- 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;
```

文档

**ST\_BandNoDataValue**, **ST\_NumBands**

## 11.8.2 ST\_SetBandIsNoData

**ST\_SetBandIsNoData** — 文档 isnodata 文档。

### Synopsis

raster **ST\_SetBandIsNoData**(raster rast, integer band=1);

文档

文档 isnodata 文档。 文档 1 文档。 文档 文档, 文档 文档 **ST\_BandIsNoData** 文档。

2.0.0 文档。

☒☒

```
-- 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
```

☒☒

[ST\\_BandNoDataValue](#), [ST\\_NumBands](#), [ST\\_SetBandNoDataValue](#), [ST\\_BandIsNoData](#)

### 11.8.3 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)

☒☒

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.

Availability: 2.5.0

☒☒

```
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
  ),
```





```
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/<br>raster/test/regress/loader/Projected.tif |
| 1     | 2       | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/<br>raster/test/regress/loader/Projected.tif |
| 1     | 3       | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/<br>raster/test/regress/loader/Projected.tif |
| 2     | 1       | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/<br>raster/test/regress/loader/Projected.tif |
| 2     | 2       | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/<br>raster/test/regress/loader/Projected.tif |
| 2     | 3       | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/<br>raster/test/regress/loader/Projected.tif |

ST\_BandMetaData, ST\_SetBandPath

## 11.9 统计函数

### 11.9.1 ST\_Count

**ST\_Count** — 返回栅格中指定值的像素数量。如果指定了 `exclude_nodata_value` 参数，则排除 NODATA 值的像素。默认情况下，`exclude_nodata_value` 为 `true`。

#### Synopsis

```
bigint ST_Count(raster rast, integer nband=1, boolean exclude_nodata_value=true);
bigint ST_Count(raster rast, boolean exclude_nodata_value);
```

参数

`nband` 指定要统计的波段。如果未指定，则统计所有波段。默认值为 1。



#### Note

`exclude_nodata_value` 参数，如果为 `true`，则排除 NODATA 值的像素。如果为 `false`，则包括 NODATA 值的像素。

2.2.0 版本中，`ST_Count(rastertable, rastercolumn, ...)` 函数被弃用。请使用 `ST_CountAgg` 函数。

2.0.0 版本中，`ST_Count` 函数被弃用。

示例

```
--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             |

相关链接

[ST\\_CountAgg](#), [ST\\_SummaryStats](#), [ST\\_SetBandNoDataValue](#)

### 11.9.2 ST\_CountAgg

**ST\_CountAgg** — 聚合函数，返回栅格中指定值的像素数量。如果指定了 `exclude_nodata_value` 参数，则排除 NODATA 值的像素。默认情况下，`exclude_nodata_value` 为 `true`。

## 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);
```



`bandwidth`: The bandwidth parameter used by the `fftshift` function. It must be a positive integer.

```
exclude_nodata_value 0000000000, 0000 nodata 00000000000000000000. 0000
00000000000000000000 exclude_nodata_value 00000000000000.
```

1. `sample_percent` is a float value between 0 and 1, representing the percentage of the dataset to sample.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.



```

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)

```



ST Count, ST SummaryStats, ST SetBandNoDataValue

### 11.9.3 ST\_Histogram

**ST\_Histogram** — (bin; ) histogram of a raster band. The histogram is returned as a table with columns min, max, count, percent, and nband.

#### 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);

The histogram is returned as a table with columns min, max, count, percent, and nband. The nband column is always 1.



#### Note

The `exclude_nodata_value` parameter is only valid when `width` is NULL. When `width` is not NULL, the histogram is calculated using the `width` values.

**width** width: The width of the histogram bins. If NULL, the width is calculated automatically. If specified, it must be an array of 9 values, width [a, b, c] [a, b, c, a, b, c, a, b, c].

Example: width [9], width [a, b, c] [a, b, c, a, b, c, a, b, c]

**bins** (breakout) : The number of bins in the histogram. If NULL, the number of bins is calculated automatically. If specified, it must be an integer value.

**right** : The right flag. If TRUE, the histogram is calculated using the right-hand side of the bins. If FALSE, the histogram is calculated using the left-hand side of the bins.

Changed: 3.1.0 Removed ST\_Histogram(table\_name, column\_name) variant.

2.0.0

Example: - 1, 2, 3

```
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: 2 bins, 6 rows.

```
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)

Example

[ST\\_Count](#), [ST\\_SummaryStats](#), [ST\\_SummaryStatsAgg](#)

### 11.9.4 ST\_Quantile

`ST_Quantile` — (population) (quantile) . , 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);

```



人口 (population) 分位数 (quantile) 百分数 (percentile)  
 25%, 50%, 75% 分位数 (percentile)



### Note

```
exclude_nodata_value 00000000, NODATA 000000000000.
```

Changed: 3.1.0 Removed ST\_Quantile(table name, column name) variant.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.



```
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   |

[ST\\_Count](#), [ST\\_SummaryStats](#), [ST\\_SummaryStatsAgg](#), [ST\\_SetBandNoDataValue](#)

### 11.9.5 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 if 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);
```

`summarystats` returns a `text` array containing the count, sum, mean, stddev, min, max for the specified band. The array is returned as `summarystats`. The array is returned as `summarystats`. The array is returned as `summarystats`.



**Note**

If `exclude_nodata_value` is `TRUE`, the nodata value is excluded from the calculation. If `exclude_nodata_value` is `FALSE`, the nodata value is included in the calculation.



**Note**

ST\_SummaryStats 函數的樣式如下。sample\_percent 為 1 時，則會計算所有像素的統計數據。

2.2.0 版本起，ST\_SummaryStats(rastertable, rastercolumn, ...) 函數已不再使用。請改用 **ST\_SummaryStatsAgg** 函數。

2.0.0 版本起，ST\_SummaryStats 函數已不再使用。

範例：計算統計數據

```
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 |

範例：計算統計數據並輸出到 CSV

PostGIS 版本 64 位的系統，需要安裝 64 位的庫文件 (例如，102,000 個 150x150 像素的影像，共 134,000 個) 需要 574 MB 的空間。

```
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 |

```
-- 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 |



summarystats, ST SummaryStatsAgg, ST Count, ST Clip

### 11.9.6 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 if 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);
```



count, sum, mean, stddev, min, max **summarystats**. **nbands** 1.



### Note

1. `nodata` 1. `exclude nodata value`



Note

sample\_percent is a float value between 0 and 1. If sample\_percent is 0, all pixels are sampled. If sample\_percent is 1, only one pixel is sampled.

2.2.0 Example

SQL

```
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)

SQL

summarystats, ST\_SummaryStats, ST\_Count, ST\_Clip

### 11.9.7 ST\_ValueCount

[illegible]

## 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, in-
teger 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 ex-
clude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, dou-
ble 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 pre-
cision roundto=0);
bigint ST_ValueCount(text rastertable, text rastercolumn, integer nband, double precision search-
value, double precision roundto=0);

```

[illegible]

```

nband = 1. searchvalues = ,
searchvalues = ,
.

```



### Note

exclude nodata value , NODATA .

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.



```
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
```



```
)
)).* AS metadata;

upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  ↔
numbands
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
          0.5 |          0.5 |    10 |    20 |     2 |     3 |     0 |     0 |    10 |  ↔
          0
```



ST\_MetaData, ST\_RastFromHexWKB, ST\_AsBinary/ST\_AsWKB, ST\_AsHexWKB

### 11.10.2 ST\_RastFromHexWKB

**ST\_RastFromHexWKB** — Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.

## Synopsis

```
raster ST_RastFromHexWKB(text wkb);
```



Given a Well-Known Binary (WKB) raster in Hex representation, return a raster.

Availability: 2.5.0



```
SELECT (ST_Metadata(
  ST_RastFromHexWKB(
    '0100000000000000000000000040000000000000008400000000000000 ↵
      E03F000000000000E03F00000000000000000000000000000000A0000000A001400'
  )
)).* AS metadata;
```

| upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | ↵ |
|------------|------------|-------|--------|--------|--------|-------|-------|------|---|
| 0.5        | 0.5        | 10    | 20     | 2      | 3      | 0     | 0     | 10   | ↵ |



ST MetaData, ST RastFromWKB, ST AsBinary/ST AsWKB, ST AsHexWKB

**11.11** ☐ ☐ ☐ ☐ ☐

### 11.11.1 ST\_AsBinary/ST\_AsWKB

ST\_AsBinary/ST\_AsWKB — Return the Well-Known Binary (WKB) representation of the raster.

## Synopsis

```
bytea ST_AsBinary(raster rast, boolean outasin=FALSE);
```

```
bytea ST_AsWKB(raster rast, boolean outasin=FALSE);
```



Returns the Binary representation of the raster. If `outasIn` is `TRUE`, out-db bands are treated as in-db. Refer to `raster/doc/RFC2-WellKnownBinaryFormat` located in the PostGIS source folder for details of the representation.

[illegible]

### Note

[illegible]

2.1.0 outasin 2.1.0.

## Enhanced: 2.5.0 Addition of ST AsWKB



```
SELECT ST_AsBinary(rast) As rastbin FROM dummy_rast WHERE rid=1;
```

rastbin

```
\001\000\000\000\000\000\000\000\000\000\000@ \000\000\000\000\000\010@ ←  
 \000\000\000\000\000\000\340? \000\000\000\000\000\340? \000\000\000\000\000\000\000\000\000\000\000\000\0
```



ST RastFromWKB, ST AsHexWKB

### 11.11.2 ST AsHexWKB

ST AsHexWKB — Return the Well-Known Binary (WKB) in Hex representation of the raster.

## Synopsis

```
bytea ST_AsHexWKB(raster rast, boolean outasin=FALSE);
```





### JPEG Output Example, multiple tiles as single raster

```
SELECT ST_AsGDALRaster(ST_Union(rast), 'JPEG', ARRAY['QUALITY=50']) As rastjpg
FROM dummy_rast
WHERE rast && ST_MakeEnvelope(10, 10, 11, 11);
```

## Using PostgreSQL Large Object Support to export raster

One way to export raster into another format is using **PostgreSQL large object export functions**. We'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;

```

**GeoTiff** ☐ ☐ ☐ ☐

```
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;
```



Section 10.3, ST GDALDrivers, ST SRID

#### 11.11.4 ST AsJPEG

ST AsJPEG — 標準 JPEG (Joint Photographic Experts Group) (標準) 形式。 1 から 3 のビット深度をサポート。 3 バイト RGB 形式。



## 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);
```



PNG(Portable Network Graphics) 格式。 使用 `ST_AsGDALRaster` 函数。 使用 3 个参数。  
 srid 指定 SRID。 使用 3 个参数。  
 使用 3 个参数。

- `nband` - 1 (PNG 1 band). `gdalinfo` shows 1 band.
- `nbands` - 4 (PNG 4 bands). `gdalinfo` shows 4 bands. `gdalinfo` shows `ARRAY[3,2,1]` 3 bands, 2 bands, 1 band.
- `compression` - 1 (PNG 1 band). `gdalinfo` shows 1 band.
- `options` - PNG `GDAL` `ST_GDALDrivers` (PNG `create_options` `GDAL`). PNG `GDAL`, `GDAL` `ZLEVEL` (PNG `GDAL` - `GDAL` 6) `GDAL`, `GDAL` `ARRAY['ZLEVEL=9']` `GDAL`. `GDAL` 2 `GDAL` `GDAL`. `GDAL` `GDAL` `GDAL` `GDAL`.

2.0.0 20200818. GDAL 1.6.0 20100522.



```
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;
```



ST AsGDALRaster, ST ColorMap, ST GDALDrivers, Section 10.3

### 11.11.6 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=sameasource);
bytea ST_AsTIFF(raster rast, text compression=", integer srid=sameasource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text compression=", integer srid=sameasource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text[] options, integer srid=sameasource);
```

## 描述

ST\_AsTIFF 将栅格数据转换为 TIFF (Tagged Image File Format) 格式。ST\_AsTIFF 函数使用 ST\_AsGDALRaster 函数将栅格数据转换为 GDAL 格式。ST\_AsGDALRaster 函数使用 SRS 信息，将栅格数据转换为 GDAL 格式。ST\_AsTIFF 函数使用 ST\_AsGDALRaster 函数将栅格数据转换为 GDAL 格式。

- **nbands** - 指定输出栅格的波段数 (PNG 支持 3 波段)。指定 RGB 波段。指定 ARRAY[3,2,1] 指定 3 波段, 指定 2 波段, 指定 1 波段。
- **compression** - 指定压缩格式: JPEG90(指定), LZW, JPEG, DEFLATE9
- **options** - GTiff 使用 GDAL 选项 (ST\_GDALDrivers 指定 GTiff 选项 create\_options 指定)。指定 GDAL 选项。
- **srid** - 指定 spatial\_ref\_sys 的 SRID 值。指定 SRID 值。

2.0.0 版本引入。GDAL 1.6.0 版本引入。

**示例: JPEG 90%**

```
SELECT ST_AsTIFF(rast, 'JPEG90') As rasttiff
FROM dummy_rast WHERE rid=2;
```

## 参数

ST\_GDALDrivers, ST\_AsGDALRaster, ST\_SRID

## 11.12 空间函数

### 11.12.1 ST\_Clip

**ST\_Clip** — Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.

#### Synopsis

```
raster ST_Clip(raster rast, integer[] nband, geometry geom, double precision[] nodataval=NULL,
boolean crop=TRUE, boolean touched=FALSE);
raster ST_Clip(raster rast, integer nband, geometry geom, double precision nodataval, boolean crop=TRUE,
boolean touched=FALSE);
raster ST_Clip(raster rast, integer nband, geometry geom, boolean crop, boolean touched=FALSE);
raster ST_Clip(raster rast, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE,
boolean touched=FALSE);
raster ST_Clip(raster rast, geometry geom, double precision nodataval, boolean crop=TRUE, boolean
touched=FALSE);
raster ST_Clip(raster rast, geometry geom, boolean crop, boolean touched=FALSE);
```



geom . , .

[illegible]

If `crop` is not specified, `true` is assumed meaning the output raster is cropped to the intersection of the `geomand` raster extents. If `crop` is set to `false`, the new raster gets the same extent as `raster`. If `touched` is set to `true`, then all pixels in the `raster` that intersect the geometry are selected.



### Note

The default behavior is `touched=false`, which will only select pixels where the center of the pixel is covered by the geometry.

Enhanced: 3.5.0 - touched argument added.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

☐☐☐☐: 2.1.0 ☐☐☐☐ C ☐☐☐☐☐☐☐☐☐☐.

Examples here use Massachusetts aerial data available on MassGIS site [MassGIS Aerial Orthos](#).

### Examples: Comparing selecting all touched vs. not all touched

```
SELECT ST_Count(rast) AS count_pixels_in_orig, ST_Count(rast_touched) AS all_touched_pixels ←
      , ST_Count(rast_not_touched) AS default_clip
FROM ST_AsRaster(ST_Letters('R'), scalex =
> 1.0, scaley =
> -1.0) AS r(rast)
      INNER JOIN ST_GeomFromText('LINESTRING(0 1, 5 6, 10 10)') AS g(geom)
ON ST_Intersects(r.rast,g.geom)
      , ST_Clip(r.rast, g.geom, touched =
> true) AS rast_touched
      , ST_Clip(r.rast, g.geom, touched =
> false) AS rast_not_touched;
```

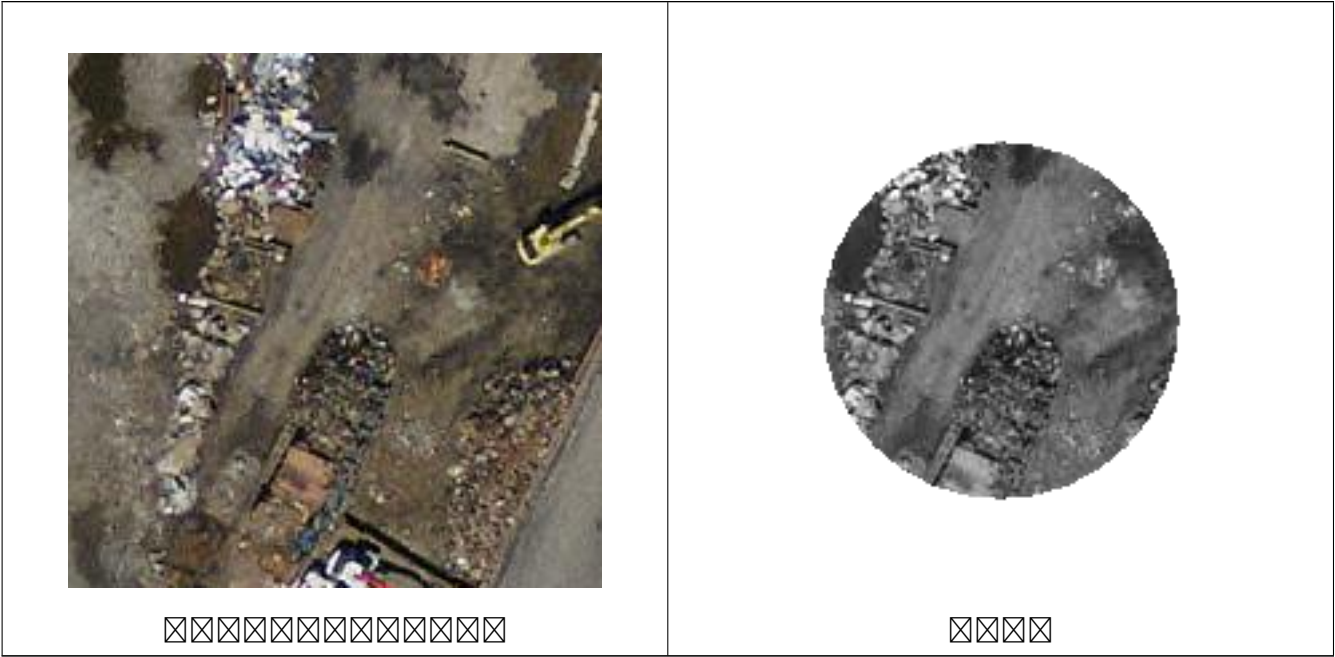
```
count_pixels_in_orig | all_touched_pixels | default_clip
-----+-----+-----
                2605 |                16 |                10
(1 row)
```

**Examples: 1 band clipping (not touched)**

```
-- 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 |



**：** **crop** 1 影像

```
-- 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;
```



SQL: 缓冲

```
-- 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_Clipping(rast,  
    ST_Buffer(ST_Centroid(ST_Envelope(rast)), 20),  
    false  
    ) from aerials.boston  
WHERE rid = 4;
```



SQL

ST\_AddBand, ST\_Count, ST\_MapAlgebra (callback function version), ST\_Intersection



## 11.12.2 ST\_ColorMap

**ST\_ColorMap** — 将栅格数据转换为 8BUI 格式 (grayscale, RGB, RGBA) 并 4 倍放大。返回 1 倍放大的栅格数据。

### 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);

参数

**rast** 栅格 nband 指定要使用的波段数。colormap 指定要使用的 8BUI 格式 4 倍放大的栅格数据。colormap 指定要使用的 8BUI 格式 8BUI 格式。

**nband** 指定要使用的波段数，默认为 1。

**colormap** 指定要使用的 8BUI 格式 colormap 格式。

**method** 指定要使用的 colormap 格式。

- grayscale 指定 grayscale - 8BUI 格式 (shades of gray) 格式
- pseudocolor - 8BUI(RGBA) 格式 4 倍放大的栅格数据，指定要使用的 colormap 格式
- fire - 8BUI(RGBA) 格式 4 倍放大的栅格数据，指定要使用的 colormap 格式
- bluered - 8BUI(RGBA) 格式 4 倍放大的栅格数据，指定要使用的 colormap 格式

**colormap** 指定要使用的 colormap 格式 (指定要使用的 colormap 格式)。指定要使用的 colormap 格式 5 倍放大的栅格数据。指定要使用的 colormap 格式，指定要使用的 colormap 格式 (RGBA) 格式 (指定要使用的 colormap 格式 0 到 255 倍放大的栅格数据)。指定要使用的 colormap 格式/指定要使用的 colormap 格式 0/100% 倍放大的栅格数据。指定要使用的 colormap 格式，指定要使用的 colormap 格式。指定要使用的 colormap 格式 NODATA 格式，指定要使用的 colormap 格式 nv, null 指定要使用的 colormap 格式。指定要使用的 colormap 格式。

```
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
```

**colormap** 指定要使用的 GDAL 格式 (color-relief) 格式 **gdaldem** 格式。

**method** 指定要使用的 colormap 格式。

- INTERPOLATE - 指定要使用的 colormap 格式。
- EXACT - 指定要使用的 colormap 格式。指定要使用的 colormap 格式 0 0 0 0(RGBA) 格式。
- NEAREST - 指定要使用的 colormap 格式。



### Note

指定要使用的 colormap 格式 **ColorBrewer** 格式。



## Warning

XXXXXXXXXXXXXXXXXXXX NODATA XXXXXXXXXXXXXXXXXXXX. NODATA XXXXXXXXXXXXXXX  
ST\_SetBandNoDataValue XXXXXXX NODATA XXXXXXXXXX.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.

[illegible]

```
-- 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;
```

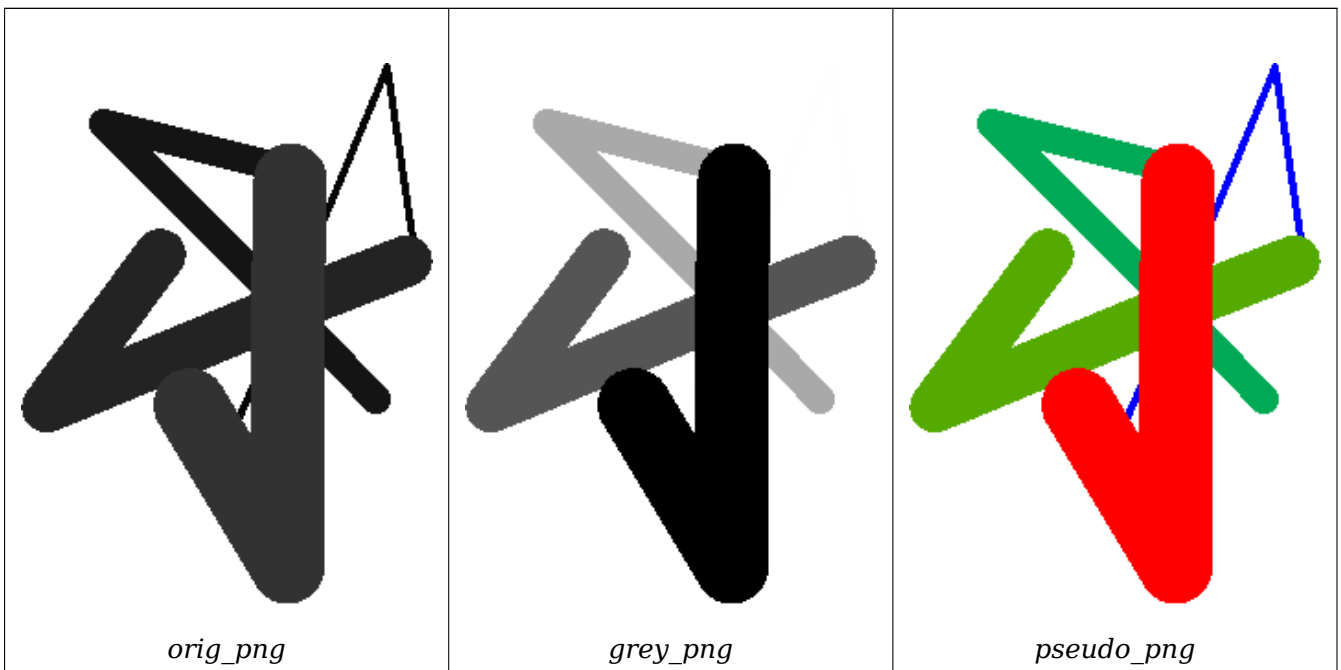
```
n_orig | ngrey | npseudo | nfire | nblooked | nred
-----+-----+-----+-----+-----+-----
      1 |      1 |      4 |      4 |      4 |      3
```

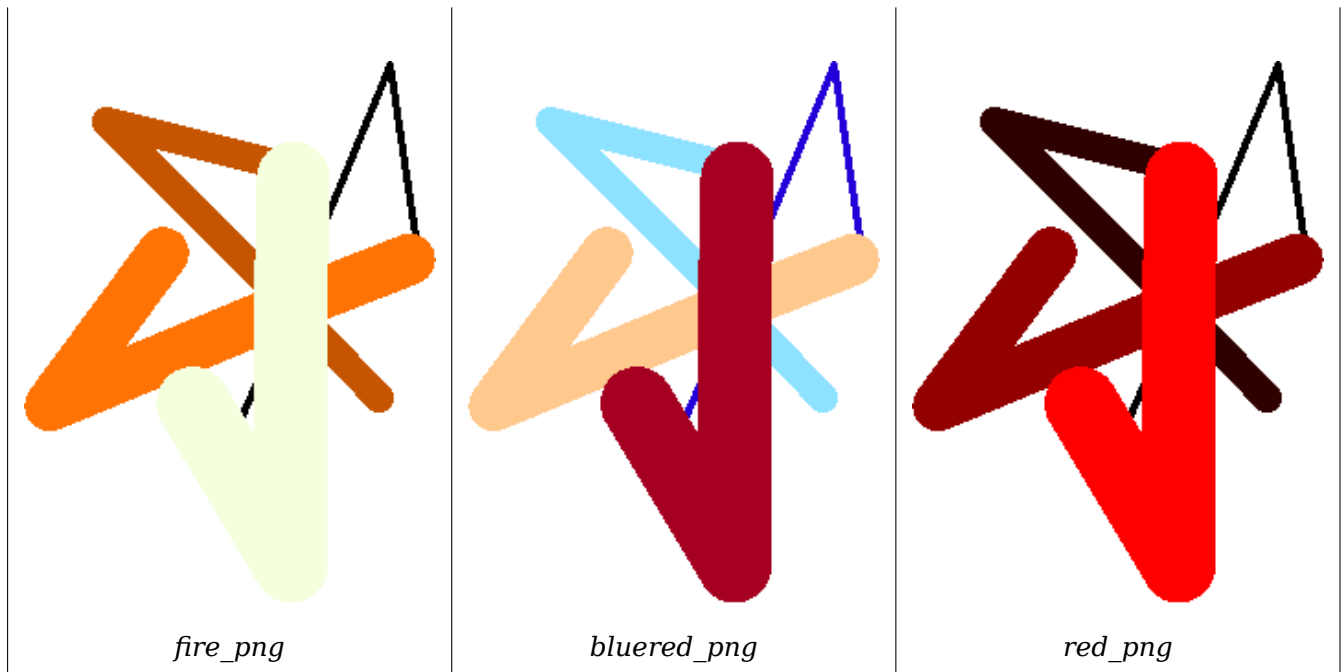
**例: ST\_AsPNG 将栅格数据转换为 PNG 图像**

```

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;

```





☒☒

[ST\\_AsPNG](#), [ST\\_AsRaster](#) [ST\\_MapAlgebra](#) (callback function version), [ST\\_Grayscale](#) [ST\\_NumBands](#), [ST\\_Reclass](#), [ST\\_SetBandNoDataValue](#), [ST\\_Union](#)

### 11.12.3 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);

☒☒

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

## 实验: 实验 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*

## 实验: 实验 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;



ST\_AsPNG, ST\_Reclass, ST\_ColorMap

#### 11.12.4 ST\_Intersection

ST Intersection —  $\{x \in \mathbb{R}^n : x \in S \text{ and } x \in T\}$ ,  $\{x \in \mathbb{R}^n : x \in S \cap T\}$ ,  $\{x \in \mathbb{R}^n : x \in S \cap T\}$ .

## Synopsis

```
setof geomval ST_Intersection(geometry geom, raster rast, integer band_num=1);
setof geomval ST_Intersection(raster rast, geometry geom);
setof geomval ST_Intersection(raster rast, integer band, geometry geomin);
raster ST_Intersection(raster rast1, raster rast2, double precision[] nodataval);
raster ST_Intersection(raster rast1, raster rast2, text returnband, double precision[] nodataval);
raster ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, double precision[]
nodataval);
raster ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, text returnband,
double precision[] nodataval);
```

[illegible]

```
geomval ST_DumpAsPolygon geomval) ST_Intersection(geometry, geometry) PostGIS
NODATA WHERE ST_Intersection
```

```
geomval ST_Intersection(rast, geom).geom' rast.val' ST_Intersection(rast, geom).geom
ST_Intersection(rast, geom).geom
```

`ST_MapAlgebraExpr`

```

XXXXXXXXXXXXXXXXXXXXX.XXXXXXXXXXXXXXXXXXXXXX.returnband XXXXXXXX
XXXXXXXXXXXX' BAND1', 'BAND2' XXX' BOTH' XXXXXXXXXX. XXXXXXXX NODATA XXX
XXXXXXXXXXXXXXXXXXXXX NODATA XXXXXXXXX. X, NODATA XXXXXXXXXXXXXXXXXXXX
NODATA XXXXXXXXXXXXXXXX.

```

```
ST_Intersection(geom1, geom2) NODATA.
ST_Band1, 'BAND2' 'BOTH' 1 2 NODATA.
nodaval[] NODATA.
NODATA, NODATA.
NODATA NODATA, ST_MinPossibleValue
NODATA. NODATA.

```

1. **ST Clip**



**Note**  
If `NODATA` is returned, `ST_MapAlgebraExpr` will return `NULL`.



**Note**  
If `ST_Clip` is used, `ST_Clip` will return `NULL`. `ST_Clip` will return `NULL` if the geometry is empty.



**Note**  
`ST_Intersects` will return `TRUE` if `ST_Intersection` returns a non-empty geometry.

PostGIS 2.0.0: 2.0.0 will return `NULL` if the geometry is empty. 2.0.0 will return `NULL` if the geometry is empty.

Example: `geomval`, `ST_Intersects` -- `geomval`

```
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                                        |     |

Example

`geomval`, `ST_Intersects`, `ST_MapAlgebraExpr`, `ST_Clip`, `ST_AsText`

### 11.12.5 ST\_MapAlgebra (callback function version)

**ST\_MapAlgebra (callback function version)** —  $\text{geometry} \times \text{integer}$  -  $\text{geometry}$ ,  $\text{integer}$ ,  $\text{geometry}$ ,  $\text{integer}$ .

## Synopsis

```

raster ST_MapAlgebra(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixelpixeltype=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 pixelpixeltype=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 pixelpixeltype=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 call-
backfunc, text pixelpixeltype=NULL, text extenttype=INTERSECTION, raster customextent=NULL, inte-
ger distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);

raster ST_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, float8[] mask, boolean
weighted, text pixelpixeltype=NULL, text extenttype=INTERSECTION, raster customextent=NULL, text[]
VARIADIC userargs=NULL);

```



□□□ 1 □□□, □□□□□, □□□□□□□□□□ 1 □□□□□□□□ 1 □□□□□□□□□□□□.

[illegible]

```
nband, nband1, nband2 [0..nband-1]. nband [0..nband-1]
[0..nband-1]. nband1 [0..nband1-1] nband2 [0..nband2-1] 2 [0..2-1] 2 [0..2-1] rast2 [0..rast2-1]
[0..rast2-1].
```

**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** mask [x][x][x][x][x][x][x][x][x] ([x][x][x][x][x][x][x][x]) [x][x] (mask [x][x][x][x][x][x][x][x][x][x]  
[x][x]) [x][x][x][x][x][x] ([x]/[x][x]) [x][x][x].

[illegible]

```
ST_BandPixelType ST_BandPixelType, ST_BandPixelType, NULL
```

**extenttype** `INTERSECTION()`, `UNION`, `FIRST(1)`, `SECOND`, `LAST`, `CUSTOM`.

**customextent** extentype ☒ CUSTOM ☒☒☒, ☒☒☒ customextent ☒☒☒☒☒☒☒☒☒. ☒☒ 1 ☒ 4  
☒☒☒☒☒☒☒☒☒☒.

**distancex** The distance in pixels from the reference cell in x direction. So width of resulting matrix would be  $2 \times \text{distancex} + 1$ . If not specified only the reference cell is considered (neighborhood of 0).

**distancey**  $\times \times \times \times \times \times$  Y  $\times \times \times \times \times \times \times \times \times \times \times \times$ .  $\times \times \times \times \times \times \times \times \times \times \times \times$  2\*distancey + 1  $\times \times \times \times$   
 $\times$ .  $\times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times$  (  $\times \times \times \times \times \times \times \times$  0 )  $\times \times \times \times \times \times$ .

[illegible]

## Note

(`VARIADIC`) `VARIADIC`, PostgreSQL `VARIADIC` `Query Language (SQL) Functions` "SQL Functions with Variable Numbers of Arguments".



Note!

### Note

```
[[{"id": "00000000-0000-0000-0000-000000000000", callbackfunc": "text"}],
```

[illegible]

$\square\square 2 \square 3 \square\square\square\square\square\square 1 \square\square\square\square\square\square\square\square\square\square\square\square. \square\square 2 \square 3 \square\square\square\square\square\square\square\square\square\square.$

□□ 4 □□□□□□□□ 1 □□□□□□□□□□ 2 □□□□□□□□□□. □□ 4 □□□□□□□□□□.

2.2.0 掩码 00000000.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.

**範例 1**

範例 1 圖, 範例 1 圖

```
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
```

範例 1 圖, 範例 1 圖

```
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
```

範例 1 圖, 範例 1 圖

```
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
```

範例 1 圖, 範例 1 圖. PostgreSQL 9.1 範例 1 圖.

```
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

```

繁體中文 PostgreSQL 9.0 繁體中文。

```

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;

```

国国: 国国 2 国 3

国国国 1 国, 国国国国

```

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

```

国国国 1 国, 国国 1 国

```

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

```

国国: 国国 4

国国国 2 国, 国国 2 国

```

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(
    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

```

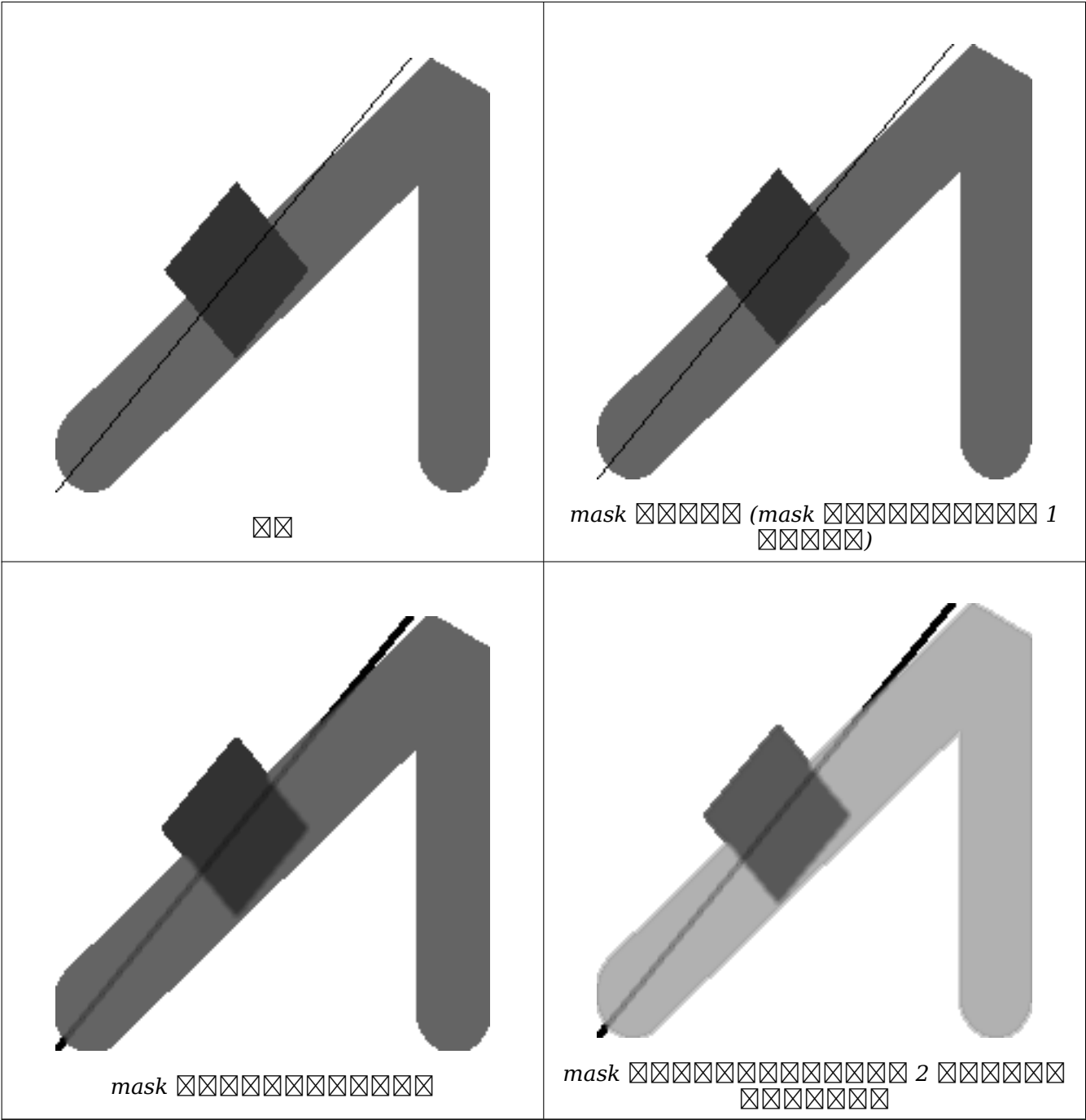
国国: **mask** 国国国国

```

WITH foo AS (SELECT
  ST_SetBandNoDataValue(
  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 other 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;

```



简体中文

`rastbandarg`, `ST_Union`, `ST_MapAlgebra` (expression version)

11.12.6 ST\_MapAlgebra (expression version)

`ST_MapAlgebra` (expression version) — 对栅格进行代数运算。1 是输入栅格，2 是输出栅格，`expression` 是 SQL 表达式。1 是输入栅格，2 是输出栅格。

## 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);

返回

返回 - 返回 1 带 2 带, 返回, 返回 SQL 返回 1 返回 1 返回。

2.1.0 返回。

返回: 返回 1, 2 (返回 1 带)

返回 (rast) 返回 expression 返回 PostgreSQL 返回, 返回 1 返回。 nband 返回, 返回 1 返回。 返回, 返回, 返回 1 返回。

pixeltype 返回, 返回。 pixeltype 返回 NULL 返回, 返回 raster 返回。

• expression 返回。

1. [rast] - 返回
2. [rast.val] - 返回
3. [rast.x] - 返回 1-返回
4. [rast.y] - 返回 1-返回

返回: 返回 3, 4 (返回 2 带)

返回 rast1, (rast2) 返回 expression 返回 2 返回, 返回 PostgreSQL 返回, 返回 1 返回。 band1, band2 返回, 返回 1 返回。 返回 (返回, 返回) 返回。 extenttype 返回。

**expression** 返回 2 返回 PostgreSQL 返回 PostgreSQL 返回/返回。 返回: (([rast1] + [rast2])/2.0)::integer

**pixeltype** 返回。 返回 **ST\_BandPixelType** 返回, 返回, NULL 返回。 返回 NULL 返回, 返回。

**extenttype** 返回

1. INTERSECTION - 返回。 返回。
2. UNION - 返回。
3. FIRST - 返回。

4. SECOND - 2つのラスタを乗算する。

**nodatalexpr** rast1 2つのラスタ NODATA 値を持つラスタ rast2 2つのラスタの乗算結果を返す。nodatalexpr は、rast2 の NODATA 値に適用される。

**nodata2expr** rast2 2つのラスタ NODATA 値を持つラスタ rast1 2つのラスタの乗算結果を返す。nodata2expr は、rast1 の NODATA 値に適用される。

**nodatanodataval** 2つのラスタ rast1 と rast2 の乗算結果を返す。NODATA 値を持つラスタは、nodatanodataval に適用される。

• expression, nodatalexpr と nodata2expr は、以下のいずれかである。

1. [rast1] - rast1 のラスタ名
2. [rast1.val] - rast1 のラスタ値
3. [rast1.x] - rast1 のラスタ X 座標
4. [rast1.y] - rast1 のラスタ Y 座標
5. [rast2] - rast2 のラスタ名
6. [rast2.val] - rast2 のラスタ値
7. [rast2.x] - rast2 のラスタ X 座標
8. [rast2.y] - rast2 のラスタ Y 座標

例: 例 1 と 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;
```

例: 例 3 と 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), 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;
```

例

**rastbandarg**, **ST\_Union**, **ST\_MapAlgebra** (callback function version)



### 11.12.7 ST\_MapAlgebraExpr

`ST_MapAlgebraExpr` — 国国国国 1 国国: 国国国国国国国国国国 PostgreSQL 国国国国国国国国国国 国国, 国国国国国国国国国国, 国国 1 国国国国国国国国国国国国国国国国. 国国国国国国国国国国国国, 国国 1 国国 国国国.

#### 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)

国国



#### Warning

**ST\_MapAlgebraExpr** 2.1.0 国国国国国国国国国国国国国国国国. 国国 **ST\_MapAlgebra (expression version)** 国国国国国国国国.

国国国国 (rast) 国国 expression 国国国国国国 PostgreSQL 国国国国国国国国国国, 国国 1 国国 国国国国国国国国国国. nband 国国国国国国国国, 国国 1 国国国国. 国国国国国国国国国国国国 国国国, 国国国国国国, 国国 1 国国国国国国国.

pixeltype 国国国国, 国国国国国国国国国国国国国国国国. pixeltype 国 NULL 国国, 国国国 国国国国 rast 国国国国国国国国国国国国国国.

国国国国国国国国国国国国国国国 [rast], 1-国国国国国国国国国国国国 [rast.x], 1-国国国国国 国国国国国国 [rast.y] 国国国国国国国国国.

2.0.0 国国国国国国国国国国.

国国

国国国国 2 国国国国国国国国 (modulo) 国国国国国国国国国 1 国国国国国国国国国国国国.

```
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 |

NODATA 0 2BUI 1

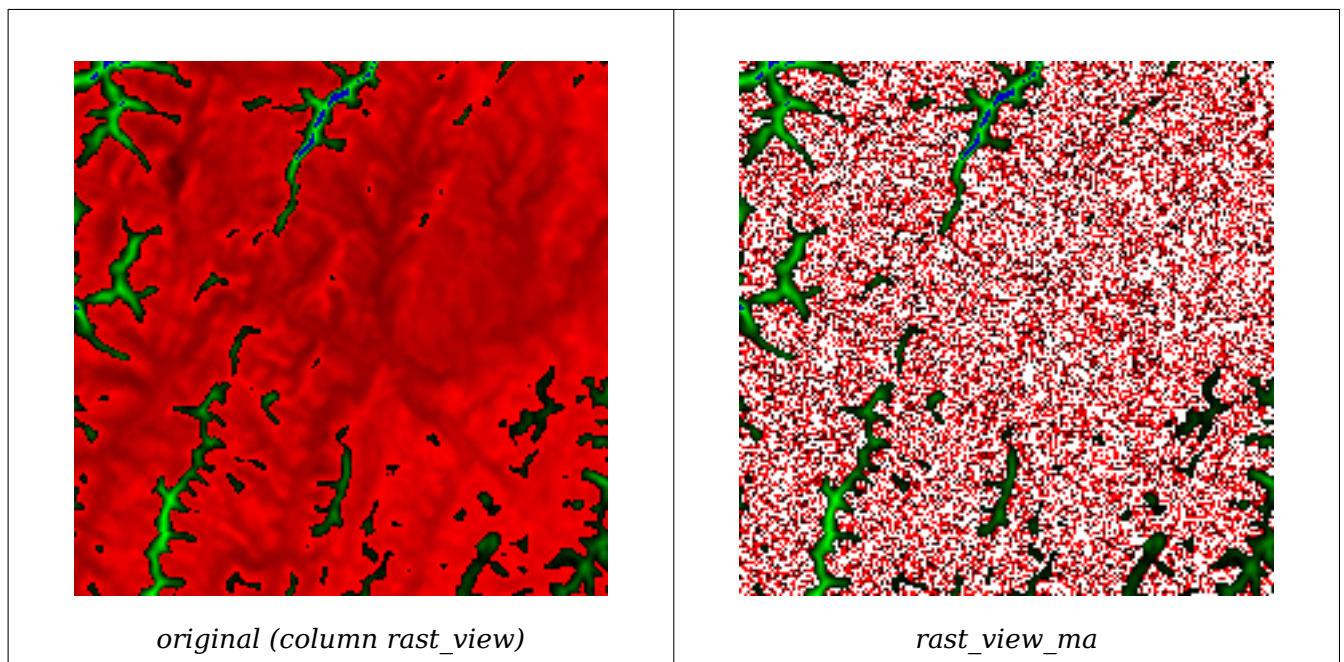
```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
UPDATE dummy_rast SET
  map_rast2 = ST_MapAlgebraExpr(rast,'2BUI'::text,'CASE WHEN [rast] BETWEEN 100 and 250
    THEN 1 WHEN [rast] = 252 THEN 2 WHEN [rast] BETWEEN 253 and 254 THEN 3 ELSE 0 END'::
    text, '0')
WHERE rid = 2;
```

```
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     |

[illegible]

```
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;
```



ST MapAlgebraExpr, ST MapAlgebraFct, ST BandPixelType, ST GeoReference, ST Value

### 11.12.8 ST\_MapAlgebraExpr

ST\_MapAlgebraExpr — 2 2: PostgreSQL 12.1, 12.2, 12.3, 12.4, 12.5, 12.6, 12.7, 12.8, 12.9, 13.0, 13.1, 13.2, 13.3, 13.4, 13.5, 13.6, 13.7, 13.8, 13.9, 14.0, 14.1, 14.2, 14.3, 14.4, 14.5, 14.6, 14.7, 14.8, 14.9, 15.0, 15.1, 15.2, 15.3, 15.4, 15.5, 15.6, 15.7, 15.8, 15.9, 16.0, 16.1, 16.2, 16.3, 16.4, 16.5, 16.6, 16.7, 16.8, 16.9, 17.0, 17.1, 17.2, 17.3, 17.4, 17.5, 17.6, 17.7, 17.8, 17.9, 18.0, 18.1, 18.2, 18.3, 18.4, 18.5, 18.6, 18.7, 18.8, 18.9, 19.0, 19.1, 19.2, 19.3, 19.4, 19.5, 19.6, 19.7, 19.8, 19.9, 20.0, 20.1, 20.2, 20.3, 20.4, 20.5, 20.6, 20.7, 20.8, 20.9, 21.0, 21.1, 21.2, 21.3, 21.4, 21.5, 21.6, 21.7, 21.8, 21.9, 22.0, 22.1, 22.2, 22.3, 22.4, 22.5, 22.6, 22.7, 22.8, 22.9, 23.0, 23.1, 23.2, 23.3, 23.4, 23.5, 23.6, 23.7, 23.8, 23.9, 24.0, 24.1, 24.2, 24.3, 24.4, 24.5, 24.6, 24.7, 24.8, 24.9, 25.0, 25.1, 25.2, 25.3, 25.4, 25.5, 25.6, 25.7, 25.8, 25.9, 26.0, 26.1, 26.2, 26.3, 26.4, 26.5, 26.6, 26.7, 26.8, 26.9, 27.0, 27.1, 27.2, 27.3, 27.4, 27.5, 27.6, 27.7, 27.8, 27.9, 28.0, 28.1, 28.2, 28.3, 28.4, 28.5, 28.6, 28.7, 28.8, 28.9, 29.0, 29.1, 29.2, 29.3, 29.4, 29.5, 29.6, 29.7, 29.8, 29.9, 30.0, 30.1, 30.2, 30.3, 30.4, 30.5, 30.6, 30.7, 30.8, 30.9, 31.0, 31.1, 31.2, 31.3, 31.4, 31.5, 31.6, 31.7, 31.8, 31.9, 32.0, 32.1, 32.2, 32.3, 32.4, 32.5, 32.6, 32.7, 32.8, 32.9, 33.0, 33.1, 33.2, 33.3, 33.4, 33.5, 33.6, 33.7, 33.8, 33.9, 34.0, 34.1, 34.2, 34.3, 34.4, 34.5, 34.6, 34.7, 34.8, 34.9, 35.0, 35.1, 35.2, 35.3, 35.4, 35.5, 35.6, 35.7, 35.8, 35.9, 36.0, 36.1, 36.2, 36.3, 36.4, 36.5, 36.6, 36.7, 36.8, 36.9, 37.0, 37.1, 37.2, 37.3, 37.4, 37.5, 37.6, 37.7, 37.8, 37.9, 38.0, 38.1, 38.2, 38.3, 38.4, 38.5, 38.6, 38.7, 38.8, 38.9, 39.0, 39.1, 39.2, 39.3, 39.4, 39.5, 39.6, 39.7, 39.8, 39.9, 40.0, 40.1, 40.2, 40.3, 40.4, 40.5, 40.6, 40.7, 40.8, 40.9, 41.0, 41.1, 41.2, 41.3, 41.4, 41.5, 41.6, 41.7, 41.8, 41.9, 42.0, 42.1, 42.2, 42.3, 42.4, 42.5, 42.6, 42.7, 42.8, 42.9, 43.0, 43.1, 43.2, 43.3, 43.4, 43.5, 43.6, 43.7, 43.8, 43.9, 44.0, 44.1, 44.2, 44.3, 44.4, 44.5, 44.6, 44.7, 44.8, 44.9, 45.0, 45.1, 45.2, 45.3, 45.4, 45.5, 45.6, 45.7, 45.8, 45.9, 46.0, 46.1, 46.2, 46.3, 46.4, 46.5, 46.6, 46.7, 46.8, 46.9, 47.0, 47.1, 47.2, 47.3, 47.4, 47.5, 47.6, 47.7, 47.8, 47.9, 48.0, 48.1, 48.2, 48.3, 48.4, 48.5, 48.6, 48.7, 48.8, 48.9, 49.0, 49.1, 49.2, 49.3, 49.4, 49.5, 49.6, 49.7, 49.8, 49.9, 50.0, 50.1, 50.2, 50.3, 50.4, 50.5, 50.6, 50.7, 50.8, 50.9, 51.0, 51.1, 51.2, 51.3, 51.4, 51.5, 51.6, 51.7, 51.8, 51.9, 52.0, 52.1, 52.2, 52.3, 52.4, 52.5, 52.6, 52.7, 52.8, 52.9, 53.0, 53.1, 53.2, 53.3, 53.4, 53.5, 53.6, 53.7, 53.8, 53.9, 54.0, 54.1, 54.2, 54.3, 54.4, 54.5, 54.6, 54.7, 54.8, 54.9, 55.0, 55.1, 55.2, 55.3, 55.4, 55.5, 55.6, 55.7, 55.8, 55.9, 56.0, 56.1, 56.2, 56.3, 56.4, 56.5, 56.6, 56.7, 56.8, 56.9, 57.0, 57.1, 57.2, 57.3, 57.4, 57.5, 57.6, 57.7, 57.8, 57.9, 58.0, 58.1, 58.2, 58.3, 58.4, 58.5, 58.6, 58.7, 58.8, 58.9, 59.0, 59.1, 59.2, 59.3, 59.4, 59.5, 59.6, 59.7, 59.8, 59.9, 60.0, 60.1, 60.2, 60.3, 60.4, 60.5, 60.6, 60.7, 60.8, 60.9, 61.0, 61.1, 61.2, 61.3, 61.4, 61.5, 61.6, 61.7, 61.8, 61.9, 62.0, 62.1, 62.2, 62.3, 62.4, 62.5, 62.6, 62.7, 62.8, 62.9, 63.0, 63.1, 63.2, 63.3, 63.4, 63.5, 63.6, 63.7, 63.8, 63.9, 64.0, 64.1, 64.2, 64.3, 64.4, 64.5, 64.6, 64.7, 64.8, 64.9, 65.0, 65.1, 65.2, 65.3, 65.4, 65.5, 65.6, 65.7, 65.8, 65.9, 66.0, 66.1, 66.2, 66.3, 66.4, 66.5, 66.6, 66.7, 66.8, 66.9, 67.0, 67.1, 67.2, 67.3, 67.4, 67.5, 67.6, 67.7, 67.8, 67.9, 68.0, 68.1, 68.2, 68.3, 68.4, 68.5, 68.6, 68.7, 68.8, 68.9, 69.0, 69.1, 69.2, 69.3, 69.4, 69.5, 69.6, 69.7, 69.8, 69.9, 70.0, 70.1, 70.2, 70.3, 70.4, 70.5, 70.6, 70.7, 70.8, 70.9, 71.0, 71.1, 71.2, 71.3, 71.4, 71.5, 71.6, 71.7, 71.8, 71.9, 72.0, 72.1, 72.2, 72.3, 72.4, 72.5, 72.6, 72.7, 72.8, 72.9, 73.0, 73.1, 73.2, 73.3, 73.4, 73.5, 73.6, 73.7, 73.8, 73.9, 74.0, 74.1, 74.2, 74.3, 74.4, 74.5, 74.6, 74.7, 74.8, 74.9, 75.0, 75.1, 75.2, 75.3, 75.4, 75.5, 75.6, 75.7, 75.8, 75.9, 76.0, 76.1, 76.2, 76.3, 76.4, 76.5, 76.6, 76.7, 76.8, 76.9, 77.0, 77.1, 77.2, 77.3, 77.4, 77.5, 77.6, 77.7, 77.8, 77.9, 78.0, 78.1, 78.2, 78.3, 78.4, 78.5, 78.6, 78.7, 78.8, 78.9, 79.0, 79.1, 79.2, 79.3, 79.4, 79.5, 79.6, 79.7, 79.8, 79.9, 80.0, 8

## Synopsis

```

raster ST_MapAlgebraExpr(raster rast1, raster rast2, text expression, text pixelttype=same_as_rast1_band
text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double preci-

```

```
sion 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);
```



## Warning

ST\_MapAlgebraExpr 2.1.0. ST\_MapAlgebra (expression version).

```

CREATE TABLE rast1, (rast2) expression 2  PostgreSQL 1
band1, band2 1
extenttype

```

**expression** `2` PostgreSQL `2` PostgreSQL  
`floor(rast1)/floor(rast2).` `floor`: (([rast1] + [rast2])/2.0)::integer

**pixeltype** 数据类型。指定 ST\_BandPixelType 数据类型，可以是 int、float、double、string、NULL 数据类型。如果为 NULL，则使用默认值。

**extenttype** ☒☒☒☒☒☒☒☒☒☒

1. INTERSECTION - . .
2. UNION - .
3. FIRST - .
4. SECOND - .

```
nodataexpr rast1 [nodata] NODATA [nodata] rast2 [nodata]
[nodata], [nodata] rast2 [nodata].
```

```
nodata2expr rast2 [ ] NODATA [ ] rast1 [ ]
[ ], [ ] rast1 [ ].
```

```
nodatanodataval 00000000 rast1 0 rast2 00000000 NODATA 000000000000
000.
```

```
pixeltype uint8_t, uint8_t *rast0, uint8_t *rast1. pixeltype int NULL uint8_t
uint8_t *rast0, uint8_t *rast1 rast1 uint8_t *rast0.
```

[[[rast1.val], [rast2.val], [rast1.x], [rast1.y]]].

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

☒☒: ☒☒ **2** ☒☒☒☒☒☒☒☒

□□□□ 2 □□□□□□□□ (modulo) □□□□□□□□□□ 1 □□□□□□□□□□□□.

```

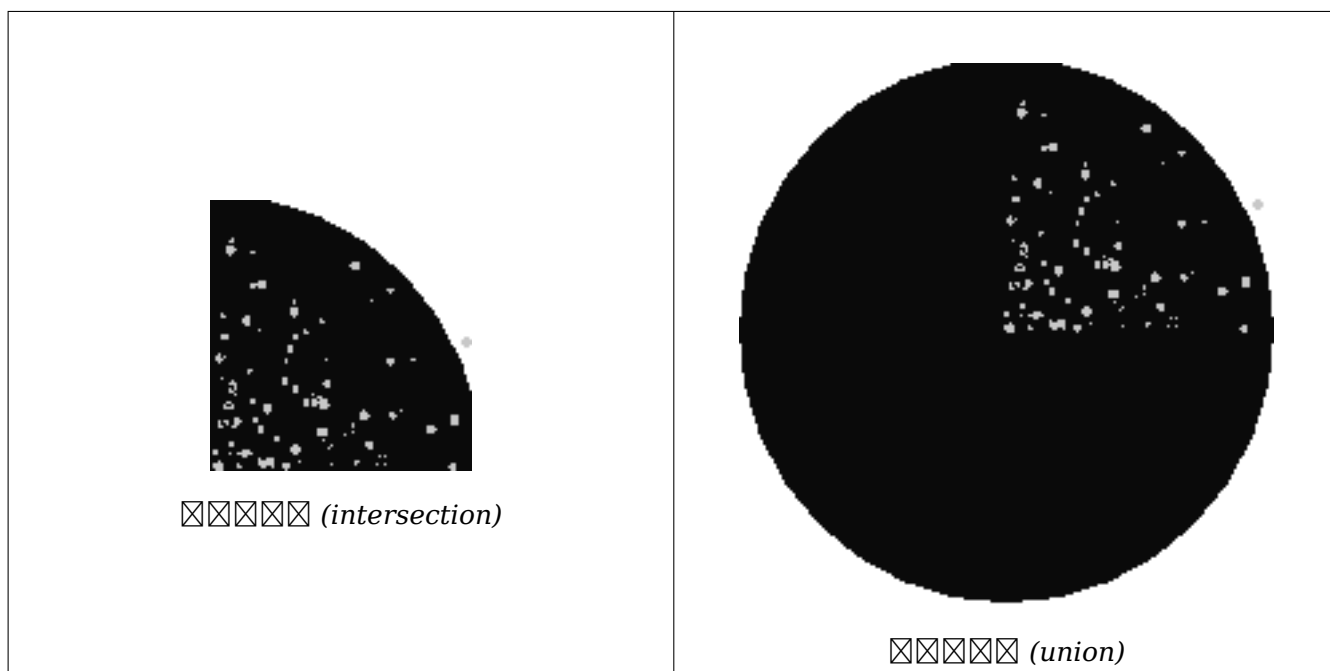
--Create a cool set of rasters --
DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

-- 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,'8BUI',0,0));

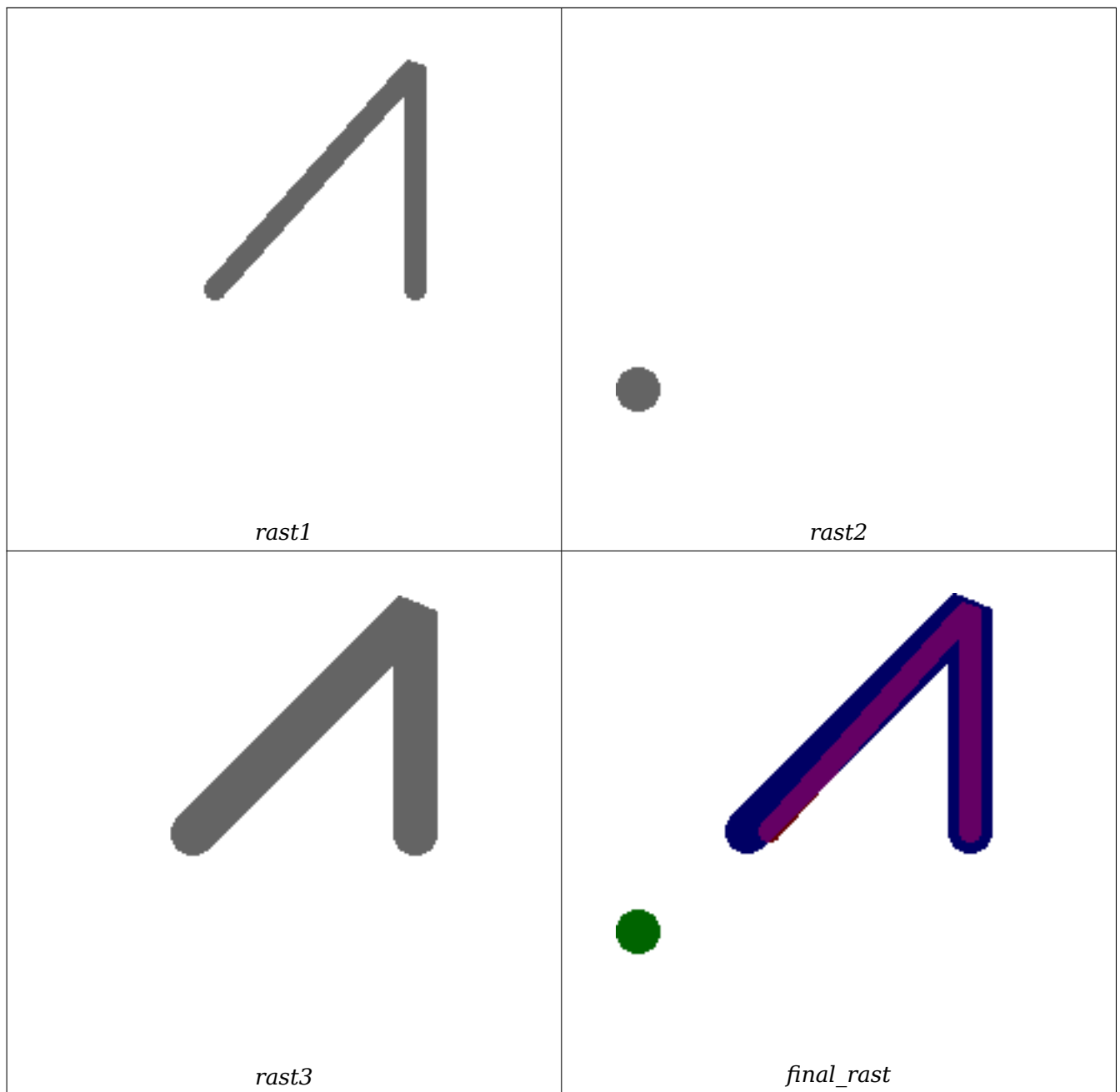
INSERT INTO fun_shapes(fun_name,rast)
WITH ref(rast) AS (SELECT rast FROM fun_shapes WHERE fun_name = 'ref' )
SELECT 'area' AS fun_name, ST_AsRaster(ST_Buffer(ST_SetSRID(ST_Point(236229, 900930),26986)
, 1000),
    ref.rast,'8BUI', 10, 0) As rast
FROM ref
UNION ALL
SELECT 'rand bubbles',
    ST_AsRaster(
        (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

```

[illegible]

```
-- 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;
```



囧囧: 囧囧囧囧囧囧囧囧囧囧囧 2 囧囧囧囧囧囧囧囧囧囧

```
-- 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;
```



图 11.12.9: 使用 ST\_MapAlgebraFct 处理栅格数据。

图 11.12.9

[ST\\_MapAlgebraExpr](#), [ST\\_AddBand](#), [ST\\_AsPNG](#), [ST\\_AsRaster](#), [ST\\_MapAlgebraFct](#), [ST\\_BandPixelType](#), [ST\\_GeoReference](#), [ST\\_Value](#), [ST\\_Union](#), [ST\\_Union](#)

### 11.12.9 ST\_MapAlgebraFct

**ST\_MapAlgebraFct** — 对栅格数据应用 1 个函数: 对 PostgreSQL 栅格数据应用 1 个函数, 对 1 个栅格数据应用 1 个函数。对 1 个栅格数据应用 1 个函数。



## 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);
```

⚠



### Warning

**ST\_MapAlgebraFct** 2.1.0 deprecated. Use **ST\_MapAlgebra** (callback function version) instead.

**rast** (raster) **onerasteruserfunc** PostgreSQL function, **band** 1-255. **args** variadic text array. **pixeltype** text, **band** 1-255. **pixeltype** NULL, **band** 1-255.

**pixeltype** text, **band** 1-255. **pixeltype** NULL, **band** 1-255. **band** 1-255. **band** 1-255.

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.

**userfunction** variadic text. **ST\_MapAlgebraFct** **userfunction**, **args**.

**Note**

(`ST_MapAlgebraFct`) VARIADIC 函数在 PostgreSQL 的 [Query Language \(SQL\) Functions](#) 中“SQL Functions with Variable Numbers of Arguments”部分有描述。

**Note**

`ST_MapAlgebraFct` 函数支持 `userfunction` 文本数组 `text[]` 参数。

2.0.0 简体中文。

SQL

创建 2 个函数 (modulo) 函数和 1 个函数。

```
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      |

创建 NODATA 函数 (0) 函数和 2BUI 函数，1 个函数。

```
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
```

```

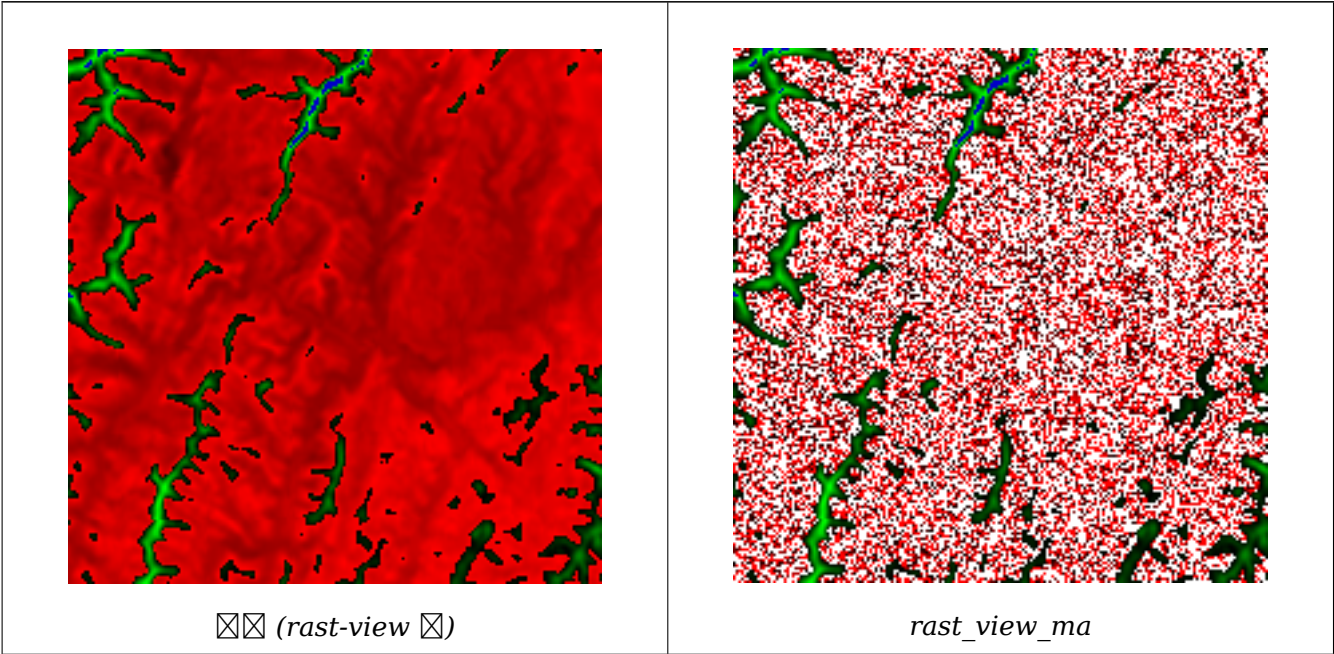
        noata := args[1];
    END IF;
    IF pixel < 251 THEN
        RETURN 1;
    ELSIF pixel = 252 THEN
        RETURN 2;
    ELSIF pixel
> 252 THEN
        RETURN 3;
    ELSE
        RETURN noata;
    END IF;
END;
$$
LANGUAGE 'plpgsql';
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ↵
[],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
```







```

    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;

```



Figure 10.1: (R: small road, G: circle, B: big road)

Figure 10.1: (R: small road, G: circle, B: big road)

```

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)';
```

[illegible]

ST\_MapAlgebraExpr, ST\_BandPixelType, ST\_GeoReference, ST\_SetValue

### 11.12.11 ST\_MapAlgebraFctNgb

ST\_MapAlgebraFctNgb — [PostgreSQL 12.1](#): [PostgreSQL](#) [Map Algebra Nearest Neighbor](#) [FCT](#). [PostgreSQL](#) [Map Algebra Nearest Neighbor](#) [FCT](#) (neighborhood) [PostgreSQL](#) [Map Algebra Nearest Neighbor](#) [FCT](#).

## Synopsis

```
raster ST_MapAlgebraFctNgb(raster rast, integer band, text pixeltypes, integer ngbwidth, integer ngbheight, regprocedure onerastngbuserfunc, text nodatamode, text[] VARIADIC args);
```



## Warning

**Warning**  
ST\_MapAlgebraFctNgb 2.1.0 .  ST\_MapAlgebra  
(callback function version) .



Figure 1: Neighborhood PostgreSQL database schema. The database schema, tables, and fields are shown.

rast □□□□□□□□□□□□□□

**band** ☐☐☐☐☐☐☐☐☐☐ (☐☐☐☐ 1)

**pixeltype** 0x000000000000. 0x00 ST\_BandPixelType 0x0000000000000000,  
0x0000, NULL 0x00000000. 0x00000000 NULL 0x00000000, rast 0x0000  
0x000000000000. 0x0000000000000000000000000000000000000000.

**ngbwidth** ☒☒ (neighborhood) ☒☒☒☒☒☒

**ngbheight** ☒☒ (neighborhood) ☒☒☒☒☒☒

**onerastngbuserfunc** ██████████ PL/pgSQL █ psql ████████.  
████████████████████ 2 ██████████.

**nodatamode** NODATA ☐☐ NULL ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐.

'ignore': XXXXXXXXXXXX NODATA XXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXXXXXXX  
XXXXX, XXXXXXXX NODATA XXXXXXXXXXXXXXXXXXXX.

```
'NULL': 000000000000 NODATA 00000000 NULL 00000000. 000000000000
00000000.
```

'value': 0x00000000 NODATA 0x000000 (0x000000000000) 0x000000. 0x00 NODATA 0x00, (0x000000000000) 'NULL' 0x00000000000000000000000000000000.

**args** ☒☒☒☒☒☒☒☒☒☒☒☒

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.



ST Rescale <https://gdal.org/user/drivers/raster/postgisraster.html>.

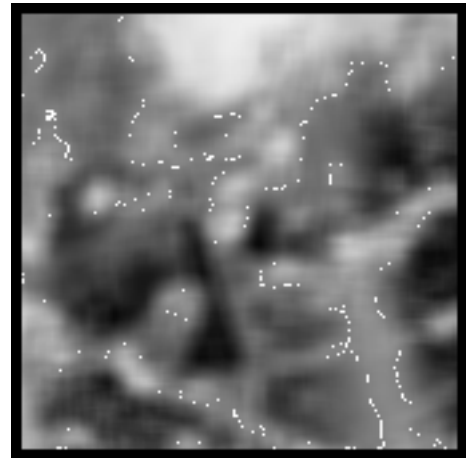
```
-- A simple 'callback' user function that averages up all the values in a neighborhood.
--
CREATE OR REPLACE FUNCTION rast_avg(matrix float[][], nodatamode text, variadic args text ↔
[])
RETURNS float AS
$$
DECLARE
    _matrix float[][];
    x1 integer;
    x2 integer;
    y1 integer;
    y2 integer;
    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;
```

```

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;

```



 4x4   
              



ST MapAlgebraFct, ST MapAlgebraExpr, ST Rescale

### 11.12.12 ST\_Reclass

ST Reclass — `ST_Reclass` is a function that takes a raster object and a reclassification table as input. The reclassification table is a data frame with two columns: the first column contains the original values (or ranges of values) and the second column contains the new values. The function returns a new raster object with the reclassified values. The `ST_Reclass` function is part of the `spatialr` package.

## Synopsis

```
raster ST_Reclass(raster rast, integer nband, text reclassexpr, text pixeltype, double precision no-
dataval=NULL);
raster ST_Reclass(raster rast, reclassarg[] VARIADIC reclassargset);
raster ST_Reclass(raster rast, text reclassexpr, text pixeltype);
```



Creates a new raster formed by applying a reclassification operation defined by the `reclassexpr` on the input raster (`rast`). Refer to [reclassarg](#) for the description of reclassification expressions. If no band is specified band 1 is assumed.

The new raster will have the same georeference, width, and height as the original raster. The bands of the new raster have pixel type of pixeltype. If reclassargset is specified then each reclassarg defines the type of the target band. Bands not designated are returned unchanged.

2.0.0

Example: Basic

2 8BUI 4BUI 101 254 NODATA

```
ALTER TABLE dummy_rast ADD COLUMN reclass_rast raster;
UPDATE dummy_rast SET reclass_rast = ST_Reclass(rast,2,'0-87:1-10, 88-100:11-15, 101-254:0-0', '4BUI',0) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast,2,i,j) As origval,
       ST_Value(reclass_rast, 2, i, j) As reclassval,
       ST_Value(reclass_rast, 2, i, j, false) As reclassval_include_nodata
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

| col | row | origval | reclassval | reclassval_include_nodata |
|-----|-----|---------|------------|---------------------------|
| 1   | 1   | 78      | 9          | 9                         |
| 2   | 1   | 98      | 14         | 14                        |
| 3   | 1   | 122     |            | 0                         |
| 1   | 2   | 96      | 14         | 14                        |
| 2   | 2   | 118     |            | 0                         |
| 3   | 2   | 180     |            | 0                         |
| 1   | 3   | 99      | 15         | 15                        |
| 2   | 3   | 112     |            | 0                         |
| 3   | 3   | 169     |            | 0                         |

1, 2, 3 1BB, 4BUI, 4BUI

```
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 |

**32BF**

32BF ((8BUI,8BUI,8BUI)

```
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)
    ]
  );
```

[ST\\_AddBand](#), [ST\\_Band](#), [ST\\_BandPixelType](#), [ST\\_MakeEmptyRaster](#), [reclassarg](#), [ST\\_Value](#)

11.12.13 ST\_ReclassExact

ST\_ReclassExact — Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.

Synopsis

raster **ST\_ReclassExact**(raster rast, double precision[] inputvalues, double precision[] outputvalues, integer bandnumber=1, text pixeltype=32BF, double precision nodatavalue=NULL);

Creates a new raster formed by applying a reclassification operation defined by the inputvalues and outputvalues arrays. Pixel values found in the input array are mapped to the corresponding value in the output array. All other pixel values are mapped to the nodatavalue.

The output pixel type defaults to float, but can be specified using the pixeltype parameter. If no bandnumber is specified band 1 is assumed.

The new raster will have the same georeference, width, and height as the original raster. Bands not designated are returned unchanged.

Availability: 3.6.0

**Example: Basic**

Create a small raster and map its pixels to new values.

```
CREATE TABLE reclassexact (
    id integer,
    rast raster
);

--
-- Create a raster with just four pixels
-- [1 2]
-- [3 4]
--
INSERT INTO reclassexact (id, rast)
SELECT 1, ST_SetValues(
    ST_AddBand(
        ST_MakeEmptyRaster(
            2,      -- width in pixels
            2,      -- height in pixels
            0,      -- upper-left x-coordinate
            0,      -- upper-left y-coordinate
            1,      -- pixel size in x-direction
            -1,     -- pixel size in y-direction (negative for north-up)
            0,      -- skew in x-direction
            0,      -- skew in y-direction
            4326    -- SRID (e.g., WGS 84)
        ),
        '32BUI'::text, -- pixel type (e.g., '32BF' for float, '8BUI' for unsigned 8-bit int)
        0.0,          -- initial value for the band (e.g., 0.0 or a no-data value)
        -99           -- nodatavalue
    ),
    1, -- band number (usually 1 for single-band rasters)
    1, -- x origin for setting values (usually 1)
    1, -- y origin for setting values (usually 1)
    ARRAY[
        ARRAY[1, 2],
        ARRAY[3, 4]
    ]::double precision[][] -- 2D array of values
);

-- Reclass the values to new values
-- and dump the values of the new raster for display
WITH rc AS (
    SELECT ST_ReclassExact(
        rast,          -- input raster
        ARRAY[4,3,2,1], -- input map
        ARRAY[14,13,12,11], -- output map
        1,             -- band number to remap
        '32BUI'        -- output raster pixtype
    ) AS rast
    FROM reclassexact
    WHERE id = 1
)
SELECT 'rce-1', (ST_DumpValues(rc.rast)).*
FROM rc;
```

☒☒

**ST\_Reclass, ST\_AddBand, ST\_Band, ST\_MakeEmptyRaster**

## 11.12.14 ST\_Union

`ST_Union` — 将 1 个或多个栅格合并为一个栅格。

### 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);
```

注意

将 1 个或多个栅格合并为一个栅格。unionarg 可以是栅格集合、uniontype 可以是 LAST(默认), FIRST, MIN, MAX, COUNT, SUM, MEAN, RANGE 等聚合函数。



#### 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.

2.0.0 版本引入。

更新: 2.1.0 版本引入 (C 语言接口)。

2.1.0 版本引入 `ST_Union(rast, unionarg)` 函数。

更新: 2.1.0 版本引入 `ST_Union(rast)` 函数 1 个参数。PostGIS 3.0 版本引入。

更新: 2.1.0 版本引入 `ST_Union(rast, uniontype)` 函数 4 个参数。

示例: 将栅格合并为一个栅格

```
-- 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;
```

示例: 将栅格合并为一个栅格

```
-- 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) ↵
);
```

**注意:** 此函数返回的栅格数据是多带的。

以下 SQL 语句创建了一个多带栅格，收集了所有与一条线相交的瓦片。

```
-- 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 aeriāls.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

返回

**unionarg, ST\_Envelope, ST\_ConvexHull, ST\_Clip, ST\_Union**

## 11.13 空间函数

### 11.13.1 ST\_Distinct4ma

ST\_Distinct4ma — 对 4 个输入栅格进行逐像素的异或运算。

#### Synopsis

float8 **ST\_Distinct4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
double precision **ST\_Distinct4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);

返回

与输入栅格具有相同尺寸和 SRID 的栅格。



#### Note

参数 1 是 **ST\_MapAlgebraFctNgb** 函数。



#### Note

参数 2 是 **ST\_MapAlgebra (callback function version)** 函数。



#### Warning

2.1.0 版本 **ST\_MapAlgebraFctNgb** 函数参数 1 是必需的。

2.0.0 版本。

返回: 2.1.0 版本 2 个参数。

例

```
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)
```

例

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra](#) (callback function version), [ST\\_Min4ma](#), [ST\\_Max4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

### 11.13.2 ST\_InvDistWeight4ma

`ST_InvDistWeight4ma` — 使用反距离加权法 (Inverse Distance Weighted method) 对栅格进行插值。

#### Synopsis

double precision **ST\_InvDistWeight4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

例

`ST_InvDistWeight4ma` (Inverse Distance Weighted method) 对栅格进行插值。

`userargs` 是包含插值参数的文本数组，至少包含 2 个参数。第一个参数是插值权重 (力率)  $k$  (范围从 0 到 1)。第二个参数是插值距离 (范围从 0 到 1)。第三个参数是插值方法 (范围从 0 到 1)。第四个参数是插值权重 (范围从 0 到 1)。第五个参数是插值距离 (范围从 0 到 1)。第六个参数是插值方法 (范围从 0 到 1)。第七个参数是插值权重 (范围从 0 到 1)。第八个参数是插值距离 (范围从 0 到 1)。第九个参数是插值方法 (范围从 0 到 1)。第十个参数是插值权重 (范围从 0 到 1)。第十一个参数是插值距离 (范围从 0 到 1)。第十二个参数是插值方法 (范围从 0 到 1)。第十三个参数是插值权重 (范围从 0 到 1)。第十四个参数是插值距离 (范围从 0 到 1)。第十五个参数是插值方法 (范围从 0 到 1)。第十六个参数是插值权重 (范围从 0 到 1)。第十七个参数是插值距离 (范围从 0 到 1)。第十八个参数是插值方法 (范围从 0 到 1)。第十九个参数是插值权重 (范围从 0 到 1)。第二十个参数是插值距离 (范围从 0 到 1)。第二十个参数是插值方法 (范围从 0 到 1)。第二十个参数是插值权重 (范围从 0 到 1)。第二十个参数是插值距离 (范围从 0 到 1)。第二十个参数是插值方法 (范围从 0 到 1)。

插值权重 (力率)  $k$  的范围从 0 到 1。

$$\hat{z}(x_o) = \frac{\sum_{j=1}^m z(x_j) d_{ij}^{-k}}{\sum_{j=1}^m d_{ij}^{-k}}$$

$k$  = 插值权重 (力率), 0 到 1 之间的值



#### Note

使用 `ST_MapAlgebra` (callback function version) 对栅格进行插值。

2.1.0 版本中，`ST_InvDistWeight4ma` 函数已弃用。



-- NEEDS EXAMPLE

ST\_MapAlgebra (callback function version), ST\_MinDist4ma

11.13.3 ST\_Max4ma

ST\_Max4ma —

Synopsis

float8 ST\_Max4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
double precision ST\_Max4ma(double precision[][] value, integer[][] pos, text[] VARIADIC userargs);

2 , userargs NODATA



**Note**  
1 ST\_MapAlgebraFctNgb



**Note**  
2 ST\_MapAlgebra (callback function version)



**Warning**  
2.1.0 ST\_MapAlgebraFctNgb 1

2.0.0  
: 2.1.0 2

```
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)
```

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra \(callback function version\)](#), [ST\\_Min4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Range4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

11.13.4 ST\_Mean4ma

ST\_Mean4ma —

Synopsis

float8 **ST\_Mean4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
double precision **ST\_Mean4ma**(double precision[][] value, integer[] pos, text[] VARIADIC userargs);

2 , userargs NODATA



Note

1 [ST\\_MapAlgebraFctNgb](#)



Note

2 [ST\\_MapAlgebra \(callback function version\)](#)



Warning

2.1.0 [ST\\_MapAlgebraFctNgb](#) 1

2.0.0

: 2.1.0 2

**☒☒: ☒☒ 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)
```

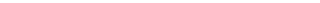
**☒☒: ☒☒ 2**

```
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)
```



ST\_MapAlgebraFctNgb, ST\_MapAlgebra (callback function version), ST\_Min4ma, ST\_Max4ma, ST\_Sum4ma  
ST\_Range4ma, ST\_StdDev4ma

### 11.13.5 ST\_Min4ma

ST Min4ma — 

## Synopsis

```
float8 ST_Min4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
double precision ST_Min4ma(double precision[][][] value, integer[] pos, text[] VARIADIC userargs);
```

[illegible]

```
00 2 000,0000 userarqs 0000 NODATA 0000000000000000.
```



**Note**

1 `ST_MapAlgebraFctNgb` is deprecated.



**Note**

2 `ST_MapAlgebra (callback function version)` is deprecated.



**Warning**

2.1.0 `ST_MapAlgebraFctNgb` is deprecated.

2.0.0 `ST_MapAlgebraFctNgb`.  
2.1.0 `ST_MapAlgebraFctNgb` is deprecated.

```
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)
```

`ST_MapAlgebraFctNgb`, `ST_MapAlgebra (callback function version)`, `ST_Max4ma`, `ST_Sum4ma`, `ST_Mean4ma`, `ST_Range4ma`, `ST_Distinct4ma`, `ST_StdDev4ma`

### 11.13.6 ST\_MinDist4ma

`ST_MinDist4ma` — Returns the minimum distance between two sets of points (polygons) in a 4D space.

#### Synopsis

double precision **ST\_MinDist4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);



**Note**

ST\_InvDistWeight4ma 是 ST\_InvDistWeight4ma 的别名。ST\_InvDistWeight4ma 是 ST\_InvDistWeight4ma 的别名。



**Note**

ST\_MapAlgebra (callback function version) 是 ST\_MapAlgebra (callback function version) 的别名。

2.1.0 是 ST\_MapAlgebra (callback function version) 的别名。

-- NEEDS EXAMPLE

ST\_MapAlgebra (callback function version), ST\_InvDistWeight4ma

### 11.13.7 ST\_Range4ma

ST\_Range4ma — 是 ST\_Range4ma 的别名。

#### Synopsis

float8 ST\_Range4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
double precision ST\_Range4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);

2 是 ST\_Range4ma 的别名。



**Note**

1 是 ST\_MapAlgebraFctNgb 的别名。



**Note**

2 是 ST\_MapAlgebra (callback function version) 的别名。



**Warning**

2.1.0 新增 `ST_MapAlgebraFctNgb` 函式，支援 1 個鄰近像素。

2.0.0 新增 `ST_MapAlgebraFctNgb`。

新增: 2.1.0 新增 `ST_MapAlgebraFctNgb` 2 個鄰近像素。

範例

```
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)
```

範例

`ST_MapAlgebraFctNgb`, `ST_MapAlgebra` (callback function version), `ST_Min4ma`, `ST_Max4ma`, `ST_Sum4ma`, `ST_Mean4ma`, `ST_Distinct4ma`, `ST_StdDev4ma`

**11.13.8 ST\_StdDev4ma**

`ST_StdDev4ma` — 計算 4 個鄰近像素的平均標準偏差。

**Synopsis**

`float8 ST_StdDev4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);`  
`double precision ST_StdDev4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);`

範例

計算 4 個鄰近像素的平均標準偏差。



**Note**

從 1 起 `ST_MapAlgebraFctNgb` 函式支援 1 個鄰近像素。

**Note**

從 2.0.0 起，`ST_MapAlgebra (callback function version)` 已成為預設行為。

**Warning**

2.1.0 起，`ST_MapAlgebraFctNgb` 已成為預設行為。

2.0.0 起，`ST_MapAlgebra` 已成為預設行為。

從 2.1.0 起，`ST_MapAlgebra` 已成為預設行為。

範例

```
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)
```

範例

`ST_MapAlgebraFctNgb`, `ST_MapAlgebra (callback function version)`, `ST_Min4ma`, `ST_Max4ma`, `ST_Sum4ma`, `ST_Mean4ma`, `ST_Distinct4ma`, `ST_StdDev4ma`

### 11.13.9 ST\_Sum4ma

`ST_Sum4ma` — 計算 4 個移動平均值的和。

#### Synopsis

```
float8 ST_Sum4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Sum4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);
```

範例

計算 4 個移動平均值的和。

從 2.0.0 起，`ST_Sum4ma` 已成為預設行為。



**Note**

1 `ST_MapAlgebraFctNgb` is deprecated.



**Note**

2 `ST_MapAlgebra (callback function version)` is deprecated.



**Warning**

2.1.0 `ST_MapAlgebraFctNgb` is deprecated.

2.0.0 `ST_MapAlgebraFctNgb`.  
2.1.0 `ST_MapAlgebraFctNgb` is deprecated.

```
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)
```

`ST_MapAlgebraFctNgb`, `ST_MapAlgebra (callback function version)`, `ST_Min4ma`, `ST_Max4ma`, `ST_Mean4ma`, `ST_Range4ma`, `ST_Distinct4ma`, `ST_StdDev4ma`

## 11.14

### 11.14.1 ST\_Aspect

`ST_Aspect` — Returns the aspect (azimuth) of a raster cell.

#### Synopsis

raster **ST\_Aspect**(raster rast, integer band=1, text pixeltype=32BF, text units=DEGREES, boolean interpolate\_nodata=FALSE);  
raster **ST\_Aspect**(raster rast, integer band, raster customextent, text pixeltype=32BF, text units=DEGREES, boolean interpolate\_nodata=FALSE);





```

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;

```

関数

**ST\_MapAlgebra (callback function version), ST\_TRI, ST\_TPI, ST\_Roughness, ST\_HillShade, ST\_Slope**

### 11.14.2 ST\_HillShade

**ST\_HillShade** — 地形陰影を生成する関数。関数の引数は、整数の帯域番号、テキストのピクセル型、二重精度の方位角、二重精度の高度、二重精度の最大明るさ、二重精度のスケール、およびbooleanのinterpolate\_nodataパラメータである。

#### 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);

関数

band 整数の帯域番号。1から255の範囲である。1はデフォルトである。255は最大値である。0はエラーである。

azimuth 方位角。0から360度の範囲である。315はデフォルトである。

altitude 高度。0から90度の範囲である。90は天頂である。45はデフォルトである。

max\_bright 最大明るさ。0から255の範囲である。255はデフォルトである。

scale 變數指定地圖縮放倍數。 例: 變數 scale=370400, 例: 變數 scale=111120 變數指定地圖縮放倍數。

interpolate\_nodata 變數指定, 變數指定 ST\_InvDistWeight4ma 變數指定 NODATA 變數指定。



### Note

變數指定, How hillshade works 變數指定。

2.0.0 變數指定。

變數指定: 2.1.0 變數指定 ST\_MapAlgebra() 變數指定, 變數指定 interpolate\_nodata 變數指定。

變數指定: 2.1.0 變數指定。 2.1.0 變數指定。

例: 例 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)
```

例: 例 2

變數指定。 變數指定 PostgreSQL 9.1 變數指定。

```
WITH foo AS (
  SELECT ST_Tile(
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
```



#### 11.14.4 ST\_Slope

$$\text{ST\_Slope} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} \cdot \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

## 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);
```

[illegible]

units . RADIANS, DEGREES() , PERCENT .

scale 0000000000000000. 00: 000000 scale=370400, 00: 000000  
scale=111120 000000.

```
interpolate_nodata ████████, ████████████████████ ST_InvDistWeight4ma ████████████████████
██████ NODATA ████████████████████████████████.
```



### Note

 (slope),  (aspect),  (hillshade)  ESRI - How hillshade works  ERDAS Field Guide - Slope Images .

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

```

XXXXXXXX: 2.1.0 XXXXXX SI_MapAlgebra() XXXXXX, XXXXXX units, scale, interpolate_nodata XXX
XXXXXXXXXXXXXXXXXXXX.

```

[illegible]

**XX: XX 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)

```

## 範例: 範例 2

範例範例範例範例範例範例範例。範例範例 PostgreSQL 9.1 範例範例範例範例範例範例。

```

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;

```

範例

**ST\_MapAlgebra (callback function version), ST\_TRI, ST\_TPI, ST\_Roughness, ST\_HillShade, ST\_Aspect**

### 11.14.5 ST\_TPI

ST\_TPI — 範例範例範例範例 (Topographic Position Index) 範例範例範例範例範例範例。

#### Synopsis

raster **ST\_TPI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate\_nodata=FALSE );



☒☒

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_Roughness](#), [ST\\_TPI](#), [ST\\_Slope](#), [ST\\_HillShade](#), [ST\\_Aspect](#)

### 11.14.7 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);

☒☒

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

☒☒

```
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')
)
```

☒☒

[ST\\_Contour](#)



### 11.14.8 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=1, double precision level_interval=100.0,
double precision level_base=0.0, double precision[] fixed_levels=ARRAY[], boolean polygonize=false);
```

**Parameters**

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.

Input parameters are:

**rast** The raster to generate the contour of

**bandnumber** The band to generate the contour of

**level\_interval** The elevation interval between contours generated

**level\_base** The "base" relative to which contour intervals are applied, this is normally zero, but could be different. To generate 10m contours at 5, 15, 25, ... the `LEVEL_BASE` would be 5.

**fixed\_levels** The elevation interval between contours generated

**polygonize** If true, contour polygons will be created, rather than polygon lines.

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

**Examples**

```
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](#)

## 11.15 国国国国国国国国

### 11.15.1 Box3D

Box3D — 国国国国国国国国国 BOX3D 国国国国国国国国.

#### Synopsis

box3d **Box3D**(raster rast);

国国

国国国国国国国国国国国国国国国国.

国国国国国国国国国国 ((MINX, MINY), (MAXX, MAXY)) 国国国国国国国国.

国国国: 2.0.0 国国国国国国 BOX3D 国 BOX2D 国国国国国国. BOX2D 国国国国国国国国国国国国, 2.0.0 国国国 BOX3D 国国国国国国.

国国

```
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) |

国国

**ST\_Envelope**

### 11.15.2 ST\_ConvexHull

ST\_ConvexHull — BandNoDataValue 国国国国国国国国国, 国国国国国国国国国国国国. 国国国国国国国国国国国国国国, ST\_Envelope 国国国国国国国国国国国国国国国国国国国国国国.

#### Synopsis

geometry **ST\_ConvexHull**(raster rast);

图

NoDataBandValue 指定, 指定. ST\_Envelope 指定.



#### Note

ST\_Envelope 指定 (floor) 指定. 指定 ST\_ConvexHull 指定.

图

PostGIS Raster Specification 文档.

```
-- Note envelope and convexhull are more or less the same
SELECT ST_AsText(ST_ConvexHull(rast)) As convhull,
       ST_AsText(ST_Envelope(rast)) As env
FROM dummy_rast WHERE rid=1;
```

```
convhull | env
-----+-----
POLYGON((0.5 0.5,20.5 0.5,20.5 60.5,0.5 60.5,0.5 0.5)) | POLYGON((0 0,20 0,20 60,0 60,0 0) ←
)
```

```
-- now we skew the raster
-- note how the convex hull and envelope are now different
SELECT ST_AsText(ST_ConvexHull(rast)) As convhull,
       ST_AsText(ST_Envelope(rast)) As env
FROM (SELECT ST_SetRotation(rast, 0.1, 0.1) As rast
      FROM dummy_rast WHERE rid=1) As foo;
```

```
convhull | env
-----+-----
POLYGON((0.5 0.5,20.5 1.5,22.5 61.5,2.5 60.5,0.5 0.5)) | POLYGON((0 0,22 0,22 61,0 61,0 0) ←
)
```

图

ST\_Envelope, ST\_MinConvexHull, ST\_ConvexHull, ST\_AsText

## 11.15.3 ST\_DumpAsPolygons

ST\_DumpAsPolygons — 指定 geomval(geom, val) 指定. 指定 1 指定.

### Synopsis

setof geomval **ST\_DumpAsPolygons**(raster rast, integer band\_num=1, boolean exclude\_nodata\_value=TRUE)

関数

`ST_DumpAsPolygon` (SRF; Set-Returning Function) 関数。 `geom` (geom) 関数 (val) 関数 `geomval` 関数 `ST_DumpAsPolygon` 関数。 `ST_DumpAsPolygon` 関数 val 関数 `ST_DumpAsPolygon` 関数。

`ST_DumpAsPolygon` 関数 `ST_DumpAsPolygon` 関数。 `ST_DumpAsPolygon` 関数 GROUP BY 関数 `ST_DumpAsPolygon` 関数。 `ST_DumpAsPolygon` 関数 `ST_DumpAsPolygon` 関数。

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

GDAL 1.7 関数。



#### 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

`ST_DumpAsPolygon` 関数 `ST_DumpAsPolygon` 関数, `ST_ValueCount` 関数。



#### Note

`ST_DumpAsPolygon` 関数 `ST_DumpAsPolygon` 関数 `ST_PixelAsPolygons` 関数。

関数

```
-- 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)) |

関数

`geomval`, `ST_AsRasterAgg`, `ST_Value`, `ST_Polygon`, `ST_ValueCount`

### 11.15.4 ST\_Envelope

ST\_Envelope — 返回给定栅格的边界框。

#### Synopsis

geometry **ST\_Envelope**(raster rast);

返回

返回的几何体具有与输入栅格相同的 SRID。如果输入栅格的 SRID 是 0，则返回的几何体的 SRID 是 0。如果输入栅格的 SRID 是 4326，则返回的几何体的 SRID 是 4326。返回的几何体的类型是 float8 类型。

返回的几何体的类型是 ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY))。

返回

```
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)) |

返回

**ST\_Envelope**, **ST\_AsText**, **ST\_SRID**

### 11.15.5 ST\_MinConvexHull

ST\_MinConvexHull — 返回给定栅格的最小凸包。

#### Synopsis

geometry **ST\_MinConvexHull**(raster rast, integer nband=NULL);

返回

返回的几何体具有与输入栅格相同的 SRID。如果输入栅格的 SRID 是 0，则返回的几何体的 SRID 是 0。如果输入栅格的 SRID 是 4326，则返回的几何体的 SRID 是 4326。返回的几何体的类型是 float8 类型。nband 是 NULL，返回的几何体的类型是 float8 类型。

2.1.0 返回的几何体的类型是 float8 类型。



```

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_1                             | mhull                               | mhull_2                             |
|---------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| POLYGON((0 0,90,9 -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)) |



ST Envelope, ST ConvexHull, ST ConvexHull, ST AsText

### 11.15.6 ST\_Polygon

ST Polygon — NODATA 

## Synopsis

geometry **ST\_Polygon**(raster rast, integer band\_num=1);

注意

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

0.1.6 添加。GDAL 1.7 添加。

更新: 2.1.0 添加 (C 添加)。添加。

更新: 2.1.0 添加, 添加。

注意

```
-- 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)))
```

```
XX
```

**ST\_Value, ST\_DumpAsPolygons**

### 11.15.7 ST\_IntersectionFractions

**ST\_IntersectionFractions** — Calculates the fraction of each raster cell that is covered by a given geometry.

#### Synopsis

```
raster ST_IntersectionFractions(raster rast, geometry geom);
```

```
XX
```

Calculates the fraction of each raster cell that is covered by a given geometry. The first argument is a raster, which defines the grid geometry to use for the calculation. The extent and cell size are read from the raster parameter. The second argument is a geometry, which is overlaid with the grid, and each grid populated based on overlaying the geometry on the grid. For polygons, the value returned for each cell is the proportion of its area that is covered by the geometry. For linestrings, the value returned for each cell is the length contained in the cell.

Availability: 3.6.0 Requires GEOS 3.14 or higher.

```
XX
```

```
CREATE TABLE raster_proportions_rast (
    name text,
    rast raster
);

INSERT INTO raster_proportions_rast (name, rast) VALUES (
    '2x2 raster covering 0,0 to 10,10',
    ST_MakeEmptyRaster(
        2, 2, -- raster width/height in pixels
        0, 10, -- upper-left corner x/y coordinates
        5, -5, -- pixel width/height in ground units
        0, 0, -- skew x/y
        0 -- SRID
    ));

--
-- This rotated square polygon covers half of each cell in the
```



```
-- raster.
--
SELECT name, ST_DumpValues(
    ST_IntersectionFractions(
        rast,
        'POLYGON((5 0, 0 5, 5 10, 10 5, 5 0))'::geometry),1)
FROM raster_proportions_rast;

2x2 raster covering 0,0 to 10,10
-----
{{0.5,0.5},{0.5,0.5}}
```

文档

## ST\_MakeEmptyRaster

## 11.16 文档

### 11.16.1 &&

&& — A 文档 B 文档 TRUE 文档.

#### Synopsis

```
boolean &&( raster A , raster B );
boolean &&( raster A , geometry B );
boolean &&( geometry B , raster A );
```

文档

&& 文档/文档 A 文档/文档 B 文档 TRUE 文档.



#### Note

文档 (operand) 文档.

2.0.0 文档.

文档

```
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
```

11.16.2 &<

&< — A  $\times$  B  $\times$  TRUE.

Synopsis

boolean &<( raster A , raster B );

&<  $\times$  A  $\times$  B  $\times$ , TRUE.



**Note**  
 (operand)  $\times$ .

```
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        |

11.16.3 &>

&> — A  $\times$  B  $\times$  TRUE.

Synopsis

boolean &>( raster A , raster B );

&>  $\times$  A  $\times$  B  $\times$ , TRUE.



**Note**

运算符 (operand) 是二进制的。

例

```
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         |

**11.16.4 =**

= — A 与 B 的交集是否为 TRUE。

**Synopsis**

```
boolean =( raster A , raster B );
```

例

= 返回 A 与 B 的交集是否为 TRUE。 PostgreSQL 支持 =, <, > 运算符 (如: GROUP BY 或 ORDER BY)。



**Caution**

运算符 (operand) 是二进制的。 如 ~ = 运算符 (group by) 不支持。

2.1.0 版本。

例

~ =

### 11.16.5 @

@ — B 与 A 的交集是否为 TRUE。返回 TRUE 或 FALSE。

#### Synopsis

```
boolean @( raster A , raster B );
boolean @( geometry A , raster B );
boolean @( raster B , geometry A );
```

返回

@ 返回 TRUE 当且仅当 B 与 A 的交集不为空。



#### Note

返回 TRUE 当且仅当 (operand) 与 A 的交集不为空。

2.0.0 添加 raster @ raster, raster @ geometry。

2.0.5 添加 geometry @ raster。

返回

~

### 11.16.6 ~=

~= — A 与 B 的交集是否为 TRUE。返回 TRUE 或 FALSE。

#### Synopsis

```
boolean ~= ( raster A , raster B );
```

返回

~= 返回 TRUE 当且仅当 A 与 B 的交集不为空。



#### Note

返回 TRUE 当且仅当 (operand) 与 B 的交集不为空。

2.0.0 添加。

图

图 2 图。

```
SELECT ST_AddBand(prec.rast, alt.rast) As new_rast
FROM prec INNER JOIN alt ON (prec.rast ~= alt.rast);
```

图

**ST\_AddBand, =**

## 11.16.7 ~

~ — A 图 B 图 TRUE 图。图。

### Synopsis

```
boolean ~( raster A , raster B );
boolean ~( geometry A , raster B );
boolean ~( raster B , geometry A );
```

图

~ 图 A 图 B 图 TRUE 图。



#### Note

图 (operand) 图。

2.0.0 图。

图

@

## 11.17 图

### 11.17.1 ST\_Contains

ST\_Contains — 图 rastA 图 rastB 图, 图 rastB 图  
图 rastA 图。

### Synopsis

```
boolean ST_Contains( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Contains( raster rastA , raster rastB );
```











**Note**

ST\_CoveredBy(geometry, ST\_Polygon(raster)) 与 ST\_CoveredBy(ST\_Polygon(raster), geometry) 不同。



**Note**

ST\_CoveredBy(geometry, ST\_Polygon(raster)) 与 ST\_CoveredBy(ST\_Polygon(raster), geometry) 不同。ST\_CoveredBy(geometry, ST\_Polygon(raster)) 返回 ST\_Polygon 是否完全覆盖。

2.1.0 版本引入。

SQL

```
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            |

SQL

**ST\_Intersects, ST\_Covers**

### 11.17.5 ST\_Disjoint

ST\_Disjoint — 判断 rastA 与 rastB 是否不相交。

**Synopsis**

boolean **ST\_Disjoint**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean **ST\_Disjoint**( raster rastA , raster rastB );

SQL

判断 rastA 与 rastB 是否不相交。如果 rastA 与 rastB 不相交，则返回 true。如果 rastA 与 rastB 相交，则返回 false。如果任一输入为 NULL，则返回 NULL。如果任一输入为 NODATA，则返回 false。



**Note**

ST\_Disjoint 与 ST\_Contains 互斥。

**Note**

`ST_Disjoint(ST_Polygon(raster), geometry)` is equivalent to `ST_Polygon`.

2.1.0

SQL

```
-- 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           |

SQL

**ST\_Intersects****11.17.6 ST\_Intersects**

`ST_Intersects` — `rastA` `rastB` `ST_Intersects`.

**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 );
```

SQL

`rastA` `rastB` `ST_Intersects`. `ST_Intersects` NULL `ST_Intersects`, `ST_Intersects`. `ST_Intersects`, `ST_Intersects` (NODATA `ST_Intersects`) `ST_Intersects`.

**Note**

ST\_Intersects(geometry, raster) 和 ST\_Intersects(raster, geometry) 是相同的。

PostGIS: 2.0.0 版本引入/支持。

**Warning**

PostGIS: 2.1.0 版本引入 ST\_Intersects(geometry, raster) 和 ST\_Intersects(raster, geometry) 的别名。

SQL

```
-- different bands of same raster
SELECT ST_Intersects(rast, 2, rast, 3) FROM dummy_rast WHERE rid = 2;

st_intersects
-----
t
```

SQL

**ST\_Intersection, ST\_Disjoint**

## 11.17.7 ST\_Overlaps

ST\_Overlaps — 返回 rasterA 和 rasterB 是否重叠的布尔值。

### Synopsis

```
boolean ST_Overlaps( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Overlaps( raster rastA , raster rastB );
```

SQL

ST\_Overlaps(rastA, rastB) 返回布尔值。如果 rastA 和 rastB 重叠，则返回 true，否则返回 false。如果任一输入为 NULL，则返回 NULL。如果任一输入为 (NODATA 值)，则返回 NODATA。

**Note**

ST\_Overlaps(geometry, raster) 和 ST\_Overlaps(raster, geometry) 是相同的。

**Note**

ST\_Overlaps(ST\_Polygon(raster), geometry) 返回 ST\_Polygon 是否重叠。

2.1.0 版本引入。

SQL

```
-- comparing different bands of same raster
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;

st_overlaps
-----
f
```

SQL

**ST\_Intersects****11.17.8 ST\_Touches**

ST\_Touches — 如果 rastA 和 rastB 在边界处接触，则返回 TRUE。

**Synopsis**

boolean **ST\_Touches**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean **ST\_Touches**( raster rastA , raster rastB );

SQL

ST\_Touches(rastA, rastB) 返回布尔值。如果 rastA 和 rastB 在边界处接触，则返回 TRUE。如果 rastA 和 rastB 没有接触，则返回 FALSE。如果任一输入为 NULL，则返回 NULL。如果任一输入为 NO\_DATA，则返回 FALSE。

**Note**

ST\_Touches 与 ST\_Overlaps 类似，但 ST\_Touches 只检查边界接触。

**Note**

ST\_Touches(ST\_Polygon(raster), geometry) 返回 ST\_Polygon 是否接触。

2.1.0 版本引入。



```
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          |



## ST\_Intersects

### 11.17.9 ST\_SameAlignment

[illegible]

## 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 );
```



§ 87(2)(b) (1, 2): (§ 87(2)(b), § 87(2)(b), SRID § 87(2)(b)) § 87(2)(b)  
§ 87(2)(b), SRID § 87(2)(b) § 87(2)(b) 4 § 87(2)(b) § 87(2)(b)  
§ 87(2)(b). § 87(2)(b) 4 § 87(2)(b) (NOTICE) § 87(2)(b)  
§ 87(2)(b).

**例 3:** 求平均数，并返回结果的小数位数。ST\_SameAlignment(  
PostgreSQL 的“聚合”函数。SUM() AVG()  
。

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

□□□□: 2.1.0 □□□□□□□□□□□□□□□□□□□□.

☒☒: ☒☒☒

```
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;
```

```
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
```

Section [10.1](#), [ST\\_NotSameAlignmentReason](#), [ST\\_MakeEmptyRaster](#)

11.17.10 ST\_NotSameAlignmentReason

ST\_NotSameAlignmentReason — Returns the reason why two rasters are not aligned.

Synopsis

text **ST\_NotSameAlignmentReason**(raster rastA, raster rastB);

Returns one of the following values, indicating the reason why the two rasters are not aligned.

**Note**

ST\_NotSameAlignmentReason returns one of the following values, indicating the reason why the two rasters are not aligned.

2.1.0 Returns the reason why the two rasters are not aligned.

```
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)
```

返回

Section 10.1, [ST\\_SameAlignment](#)

### 11.17.11 ST\_Within

**ST\_Within** — 返回 rasterB 是否完全包含在 rasterA 中, 返回 rasterA 是否完全包含在 rasterB 中。

#### Synopsis

boolean **ST\_Within**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean **ST\_Within**( raster rastA , raster rastB );

返回

返回 rasterB 是否完全包含在 rasterA 中, 返回 rasterA 是否完全包含在 rasterB 中。如果 rasterA 与 rasterB 的波段数不同, 则返回 NULL。如果 rasterA 或 rasterB 包含任何 NULL 值, 则返回 NULL。如果 rasterA 或 rasterB 包含任何 (NODATA 值) 则返回 NULL。



**Note**  
返回 (operand) 返回。



**Note**  
返回 ST\_Within(ST\_Polygon(raster), geometry) 与 ST\_Within(geometry, ST\_Polygon(raster)) 返回 ST\_Polygon 是否包含。



**Note**  
ST\_Within() 与 ST\_Contains() 返回。返回, ST\_Within(rastA, rastB) 返回 ST\_Contains(rastB, rastA) 返回。

2.1.0 返回。

返回

```
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
```





### 11.17.13 ST\_DFullyWithin

ST\_DFullyWithin — rasterA rasterB

#### 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 );

rasterA rasterB. NULL, (NODATA) SRID.



**Note**  
(operand)



**Note**  
ST\_DFullyWithin(ST\_Polygon(raster), geometry) ST\_Polygon

2.1.0

```
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               |

ST\_Within, ST\_DWithin

## 11.18 Raster Tips

### 11.18.1 Out-DB Rasters

#### 11.18.1.1 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'
```

#### 11.18.1.2 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

### 11.18.1.2.1 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
```

### 11.18.1.2.2 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 12

# PostGIS Extras

This chapter documents features found in the extras folder of the PostGIS source tarballs and source repository. These are not always packaged with PostGIS binary releases, but are usually PL/pgSQL based or standard shell scripts that can be run as is.

### 12.1 address\_standardizer

**PAGC standardizer** (fork) (PAGC PostgreSQL **lexicon**).

**lexicon** (lex) **gazetteer** (gaz) **lexicon**.

```
CREATE EXTENSION address_standardizer;
-- address_standardizer PostgreSQL extension,
-- address_standardizer_data_us US address standardizer data,
-- CREATE EXTENSION address_standardizer_data_us;
--
```

PostGIS extensions/address\_standardizer **lexicon**, **gazetteer**.

**lexicon** Section 2.3 **lexicon**.

#### 12.1.1 address\_standardizer\_data\_us

**lexicon** (lex) **gazetteer** (gaz) **lexicon** (macro) **lexicon**, **lexicon** (micro) **lexicon**.

**lexicon** **lexicon**/, **lexicon**.

**lexicon**/ (zip code) (Perl) **lexicon**. **lexicon** **lexicon** **lexicon**.

**lexicon** (Perl) **lexicon**. **lexicon** **lexicon** **lexicon**.

## 12.1.2 国国国国国国国国国

### 12.1.2.1 stdaddr

**stdaddr** — 国国国国国国国国国国国国国国国. **standardize\_address** 国国国国国国国国国国国.

国国

国国国国国国国国国国国国国国国. **standardize\_address** 国国国国国国国国国国国. **PAGC Postal Attributes** 国国国国国国国国国国国国国国国.

**rules table** 国国国国国国国国国国国国国国国国国国国国国国国.



This method needs **address\_standardizer** extension.

**building** 国国 (国国国 0) 国国: 国国国国国国国国国国国国国国国. 国国国国国国国国国国国国国国国. 国国国国国国国国国国国国国国国.

**house\_num** 国国 (国国国 1) 国国: 国国国国国国国国. 国: 75 State Street 国 75 国国

**predir** 国国 (国国国 2) 国国: North, South, East, West 国国国国国国国国国国国国国国国 (STREET NAME PRE-DIRECTIONAL) 国国.

**qual** 国国 (国国国 3) 国国: 国国国国国国国 (STREET NAME PRE-MODIFIER) 国国. 国: 3715 OLD HIGHWAY 99 国国 *OLD*

**pretype** 国国 (国国国 4) 国国: 国国国国国国 (STREET PREFIX TYPE) 国国.

**name** 国国 (国国国 5) 国国: 国国 (STREET NAME) 国国.

**suftype** 国国 (国国国 6) 国国: St, Ave, Cir 国国国国国国国国国 (STREET POST TYPE) 国国. 国国 国国国国国国国国国国国国国国国. 国: 75 State Street 国国 *STREET*

**sufdir** 国国 (国国国 7) 国国: 国国国国国国国国国国国 (STREET POST-DIRECTIONAL) 国国. 国 国国国国国国国国国国国国国国国国国. 国: 3715 TENTH AVENUE WEST 国国 *WEST*

**ruralroute** is text (token number 8): **RURAL ROUTE** . Example 7 in RR 7.

**extra** 国国国国国: 国国国国国国国国国国.

**city** 国国 (国国国 10) 国国: 国: 国国国

**state** 国国 (国国国 11) 国国: 国: 国国国国国

**country** 国国 (国国国 12) 国国: 国: USA

**postcode** 国国 (国国国 13) 国国国 (postal code, zip code) 国国: 国: 02109

**box** 国国 (国国国 14, 15) 国国国 (POSTAL BOX NUMBER) 国国: 国: 02109

**unit** 国国 (国国国 17) 国国国国国国国国国国国: 国: APT 3B 国国 *3B*

## 12.1.3 国国国国国国国国国

### 12.1.3.1 rules table

**rules table** — 国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国国. 国国国国国国国, 国国 -1(国国; terminator), 国国国国国国国国国国, 国国 -1, 国国国国国国国国国国国国国, 国国国国国国国国国国国国国国国国国国国国国国国.

。

，。

**id** 。

**rule** 。

， -1, -1, 。

TYPE NUMBER TYPE  
DIRECT QUALIF , STREET STREET SUFTYP SUFDIR QUALIF  
。 6 ARC\_C 。

**stdaddr**。

。

-1 。

。

**AMPERS** (13)。 (&) "and" 。

**DASH** (9)。 (句讀法; punctuation) 。

**DOUBLE** (21)。 2 。

**FRACT** (25)。

**MIXED** (23)。

**NUMBER** (0)。

**ORD** (15)。 "First" "1st" 。

**ORD** (18)。

**WORD** (1)。

。

**BOXH** (14)。

**BUILDH** (19)。

**BUILD** 。

**DIRECT** (22)。

**MILE** (20)。

**ROAD** (6)。

**RR** (8)。

**TYPE** (2)。







### 12.1.3.3 gaz table

gaz table —  (gaz) ,  (1)  (the section called “”)   (2) .



A gaz (short for gazeteer) table is used to standardize place names and associate that input with the section called “**地名**” and (b) standardized representations. For example if you are in US, you may load these with State Names and associated abbreviations.

[illegible]

id    □□□□□□□□

**seq** : ? -

**word** ☐☐☐: ☐☐☐☐

**stdword** □□□: □□□□□□□□

**token** :                                                                                                                                             

### **12.1.4**

#### 12.1.4.1 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);
```



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.



address\_standardizer\_data\_us 

```
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) |          |            |             |                   |              |



**stdaddr**, **rules table**, **lex table**, **gaz table**, **Pagc\_Normalize\_Address**

#### 12.1.4.2 parse\_address

```
parse_address — .
```

## Synopsis

```
record parse_address(text address);
```



Returns takes an address as input, and returns a record output consisting of fields *num*, *street*, *street2*, *address1*, *city*, *state*, *zip*, *zipplus*, *country*.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.

✔ This method needs address\_standardizer extension.



```
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    |

☐ ☐ ☐ ☐ ☐

```
-- 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 | 323     |
| 2        | 77  | Massachusetts Avenue | Cambridge | MA    | 02139 |         |
| 3        | 25  | Wizard of Oz         | Walaford  | KS    | 99912 |         |
| 4        | 26  | Capen Street         | Medford   | MA    |       |         |
| 5        | 124 | Mount Auburn St      | Cambridge | MA    | 02138 |         |
| 6        | 950 | Main Street          | Worcester | MA    | 01610 |         |
| (6 rows) |     |                      |           |       |       |         |



| 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)

例 2: 標準化地址。

```
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)

例

`stdaddr`, `rules table`, `lex table`, `gaz table`, `Pagc_Normalize_Address`

## 12.2 TIGER 繁體中文

TIGER 是美國人口普查局 (US Census Bureau) 提供的一個數據庫，PostGIS 可以通過 `tiger` 擴展來使用它。

- **Nominatim** is an OpenStreetMap database. It uses the `osm2pgsql` tool, which requires PostgreSQL 8.4 or later and PostGIS 1.5 or later. Nominatim is a geocoding and reverse geocoding service. TIGER is a US Census Bureau database. Nominatim uses TIGER data to provide US address data. SQL is used to query the database.
- **GIS Graphy** is a PostGIS extension. It uses OSM (OpenStreetMap) data. OSM is a free, editable map of the world. Nominatim is a geocoding and reverse geocoding service. Nominatim uses TIGER data to provide US address data. Java 1.5, Servlet apps, Solr are used. GIS Graphy is a web application.

### 12.2.1 Drop\_Indexes\_Generate\_Script

Drop Indexes Generate Script — TIGER data. This script drops all indexes on the `tiger_data` schema.

#### Synopsis

text **Drop\_Indexes\_Generate\_Script**(text param\_schema=tiger\_data);

Example

TIGER data. This script drops all indexes on the `tiger_data` schema.

(bloat) is a common problem. **Install Missing Indexes** script can be used to install missing indexes.

2.0.0 version.

Example

```
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;
:
:

```

❏❏

[Install\\_Missing\\_Indexes, Missing\\_Indexes\\_Generate\\_Script](#)

## 12.2.2 Drop\_Nation\_Tables\_Generate\_Script

Drop\_Nation\_Tables\_Generate\_Script — 删除 tiger\_data 数据库 county\_all, state\_all 表并生成 tiger\_data 数据库 county, state 表 (州) 的索引。

### Synopsis

text **Drop\_Nation\_Tables\_Generate\_Script**(text param\_schema=tiger\_data);

❏❏

删除 tiger\_data 数据库 county\_all, state\_all 表并生成 tiger\_data 数据库 county, state 表 (州) 的索引。tiger\_2010 和 tiger\_2011 表的索引将不会被删除。

2.1.0 版本引入。

❏❏

```

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;

```

❏❏

[Loader\\_Generate\\_Nation\\_Script](#)

## 12.2.3 Drop\_State\_Tables\_Generate\_Script

Drop\_State\_Tables\_Generate\_Script — 删除 tiger\_data 数据库 (州) 的 county\_all, state\_all 表并生成 tiger\_data 数据库的 county, state 表。



## Synopsis

text **Drop\_State\_Tables\_Generate\_Script**(text param\_state, text param\_schema=tiger\_data);

說明

Drop\_State\_Tables\_Generate\_Script (州) 會生成一個 SQL 腳本，用於刪除 tiger\_data 資料庫中的所有表。該腳本會生成一個 SQL 腳本，用於刪除 tiger\_data 資料庫中的所有表。該腳本會生成一個 SQL 腳本，用於刪除 tiger\_data 資料庫中的所有表。該腳本會生成一個 SQL 腳本，用於刪除 tiger\_data 資料庫中的所有表。

2.0.0 版本新增。

說明

```
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;
```

說明

**Loader\_Generate\_Script**

## 12.2.4 Geocode

Geocode — 將地址 (字符串) 轉換為 NAD83 坐標，並返回一個包含坐標、地址、評分等的記錄集。該函數會根據提供的地址生成一個 SQL 腳本，用於刪除 tiger\_data 資料庫中的所有表。該腳本會生成一個 SQL 腳本，用於刪除 tiger\_data 資料庫中的所有表。該腳本會生成一個 SQL 腳本，用於刪除 tiger\_data 資料庫中的所有表。該腳本會生成一個 SQL 腳本，用於刪除 tiger\_data 資料庫中的所有表。

## 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);

## 安裝

PostGIS (PostgreSQL 空間擴展) 需要 NAD83 座標系，並需要 normalized\_address (addy) 欄位。TIGER 數據 (edge, face, addr), PostgreSQL (soundex, levenshtein), 以及 PostGIS 需要 TIGER 數據。安裝時，請指定 (x/y/z) 為 10 位元。

安裝: 2.0.0 需要 TIGER 2010 數據，並需要 normalized\_address 欄位。安裝時，請指定 (x/y/z) 為 10 位元。max\_results 欄位。

## 安裝

PostGIS (MA), PostGIS (MN), PostGIS (CA), PostGIS (RI) 需要 TIGER 數據。PostgreSQL 9.1rc1/PostGIS 2.0 需要 3.0 GHz 處理器，2GB 記憶體，7 個磁碟。

安裝時，請指定 (61 位元)。

```
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
```

安裝時，請指定 (122 ~ 150 位元)。

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
       (addy).address As stno, (addy).streetname As street,
       (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,( ←
       addy).zip
FROM geocode('226 Hanover Street, Boston, MA',1) As g;
rating |          wktlonlat          | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----
1 | POINT(-71.05528 42.36316) | 226 | Hanover | St | Boston | MA | 02113
```

安裝時，請指定 (500 位元)。

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
       (addy).address As stno, (addy).streetname As street,
       (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,( ←
       addy).zip
FROM geocode('31 - 37 Stewart Street, Boston, MA 02116',1) As g;
rating |          wktlonlat          | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----
70 | POINT(-71.06466 42.35114) | 31 | Stuart | St | Boston | MA | 02116
```

安裝時，請指定 (batch) 欄位。max\_results = 1 欄位。安裝時，請指定 (x/y/z) 為 10 位元。

```
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, ↵
      2 | 77 Massachusetts Avenue, Cambridge, MA 02139 | -71.09396 | 42.35961 | 77 ↵
      3 | 25 Wizard of Oz, Walaford, KS 99912323 | -97.92913 | 38.12717 | Willowbrook, ↵
      108
(3 rows)

```

⌘: ⌘⌘⌘⌘⌘⌘

```

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



Geocode, Pprint\_Addy, ST\_AsText

12.2.6 Get\_Geocode\_Setting

Get\_Geocode\_Setting — tiger.geocode\_settings

Synopsis

text **Get\_Geocode\_Setting**(text setting\_name);

tiger.geocode\_settings. The following table lists the settings and their default values. The settings are listed in the order they appear in the tiger.geocode\_settings table.

| 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_pgc_address_parser         | false   | boolean | normalize | ↔ | If set to true, will try to use the address_standardizer extension (via pgc_normalize_address) instead of tiger normalize_address built one |

2.2.0 geocode\_settings\_default. The following table lists the settings and their default values. The settings are listed in the order they appear in the tiger.geocode\_settings table.

2.1.0.

:

```
SELECT get_geocode_setting('debug_geocode_address') As result;
result
-----
false
```



## 12.2.8 Install\_Missing\_Indexes

`Install_Missing_Indexes` — 为指定的表 (join) 和列 (key) 安装缺失的索引。

### Synopsis

boolean **Install\_Missing\_Indexes**();

描述

`tiger` 和 `tiger_data` 数据库包含用于生成索引的脚本，这些脚本使用 SQL DDL 语句，为指定的表安装缺失的索引。这些脚本位于 `Missing_Indexes_Generate_Script` 文件中，位于 `update_geocode.sql` 文件中。2.0.0 版本包含这些脚本。

用法

```
SELECT install_missing_indexes();
       install_missing_indexes
-----
t
```

描述

[Loader\\_Generate\\_Script](#), [Missing\\_Indexes\\_Generate\\_Script](#)

## 12.2.9 Loader\_Generate\_Census\_Script

`Loader_Generate_Census_Script` — 为指定的表 (州) 安装 TIGER 索引 (州) 的 tract, bg, tabblocks 索引。该脚本位于 `tiger_data` 数据库中。

### Synopsis

setof text **loader\_generate\_census\_script**(text[] param\_states, text os);

描述

该脚本为指定的表 (州) 安装 TIGER 索引 (州) 的 tract, bg, tabblocks 索引。该脚本位于 `tiger_data` 数据库中。

该脚本使用 `unzip` 命令 (7-zip 包) 解压，并使用 `wget` 命令。该脚本位于 [Section 4.7.2](#) 中。

该脚本使用 OS 命令。

1. `loader_variables` - 设置, 设置, 设置 (staging) 设置。
2. `loader_platform` - 设置。 设置。
3. `loader_lookuptables` - 设置 (州, 县), 设置, 设置。 设置。 设置, 设置, 设置, 设置。 设置 (州名) 设置, TIGER 设置。 设置: `tiger.faces` 设置 `tiger_data.ma_faces` 设置。

2.0.0 设置。



#### Note

**Loader\_Generate\_Script** 设置, PostGIS 2.0.0 alpha5 设置 TIGER 设置。 设置, 设置 (州) 设置。

设置

设置。

```
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');"
:
```



```
.sh 简体中文.
```

```
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
:
:
```

❏

## Loader\_Generate\_Script

### 12.2.10 Loader\_Generate\_Script

Loader\_Generate\_Script — 简体中文 (州) ❏, TIGER ❏ tiger\_data ❏. ❏ (州) ❏. TIGER 2010 ❏, ❏, ❏, ❏.

#### Synopsis

```
setof text loader_generate_script(text[] param_states, text os);
```

❏

❏ (州) ❏, TIGER ❏ tiger\_data ❏. ❏ (州) ❏.

❏ unzip ❏ (❏ 7-zip ❏) ❏, ❏ wget ❏. ❏ Section 4.7.2 ❏. ❏ (州) ❏. ❏ "staging" ❏ "temp" ❏.

❏ OS ❏.

1. loader\_variables - ❏, ❏, ❏ (staging) ❏.

2. loader\_platform - 加载平台信息。返回平台名称、版本、架构、操作系统、CPU 架构、内存大小、磁盘大小、网络接口、GPU 信息、虚拟化信息、容器信息、安全信息、日志配置、性能监控配置、备份配置、恢复配置、升级配置、部署配置、配置项、配置值、配置描述、配置来源、配置时间、配置人、配置组、配置项、配置值、配置描述、配置来源、配置时间、配置人、配置组。
3. loader\_lookuptables - 加载查找表信息。返回查找表名称、表名、表结构、表数据、表索引、表权限、表备注、表来源、表时间、表人、表组、表名、表结构、表数据、表索引、表权限、表备注、表来源、表时间、表人、表组。

2.0.0 . TIGER 2010  (tract),  (bg),  (tabblock) .



### 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.

11

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;
```

[illegible]

```
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
:
:
```

.sh [redacted].

```
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 PGPOR=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}/*.*
:
:
```



Section 2.4.1, Pprint Addy, ST\_AsText

### 12.2.11 Loader\_Generate\_Nation\_Script

Loader\_Generate\_Nation\_Script — 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000.

## Synopsis

```
text loader generate nation script(text os);
```



```

tiger_data %>% county_all, county_all_lookup, state_all %>%
  select(county, county_lookup, state).

```

unzip (7-zip) , wget .  
Section 4.7.2 .

```

OS tiger.loader_platform,tiger.loader_var
tiger.loader lookuptables

```

1. loader\_variables - 加载变量 (staging) 变量。  
变量。
2. loader\_platform - 加载平台。 加载平台。  
/。 加载平台。
3. loader\_lookuptables - 加载查找表 (表, 表), 加载查找表, 加载查找表。  
加载查找表。 加载查找表, 加载查找表, 加载查找表。  
加载查找表 (州名) 表, TIGER 加载查找表。  
加载查找表。 表: tiger.faces 加载查找表 tiger data.ma faces 加载查找表。

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.

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

**Note!**

tiger\_2010 (州) tiger\_2011 ,  
Drop\_Nation\_Tables\_Generate\_Script  
.

[illegible]

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

□□□/□□□□□□□□□□□□□□□□□□□□□□□□.

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



## Loader Generate Script, Missing Indexes Generate Script

### 12.2.12 Missing\_Indexes\_Generate\_Script

[illegible]

## Synopsis

```
text Missing_Indexes_Generate_Script();
```



tiger < tiger\_data <>>> (join) <>>> (key) <>>><br>  
SQL DDL <>>>. <>>><br>, <>>><br>, <>>><br>, <>>>.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.



```
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);
```



## Loader\_Generate\_Script, Install\_Missing\_Indexes

### 12.2.13 Normalize\_Address

**Normalize\_Address** — `uint8_t norm_addy`, `tiger_geocoder` (TIGER `uint8_t`)

## Synopsis

```
norm addy normalize_address(varchar in address);
```

[illegible]

tiger tiger geocoder (州)/  
TIGER  
.tiger

tiger

```

XXXXXXXX norm_addy XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX. XX () XXXXXXXXXXXXXXXXXXXX
XXXXXXXX. [] XXXXXXXXXXXXXXXXXXXX:

```

```
(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [state-Abbrev] [zip] [parsed] [zip4] [address alphanumeric]
```

Enhanced: 2.4.0 norm addy object includes additional fields zip4 and address alphanumeric.

1. address : .
2. **predirAbbrev**  **varchar** : N, S, E, W . **direction\_look** .



¶¶

¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶  
norm\_addy ¶¶¶¶¶¶¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶  
¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

¶¶¶¶ tiger ¶¶¶¶¶¶ tiger\_geocoder ¶¶¶¶¶¶¶¶¶¶¶¶ page\_\* ¶¶¶¶¶¶¶¶¶¶¶¶. ¶  
¶¶¶¶¶¶¶¶¶¶¶¶ TIGER ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. tiger  
¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶ tiger ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

¶¶¶¶ norm\_addy ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. ¶¶ () ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶  
¶¶¶¶, [] ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶:

**Normalize\_Address** ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

2.1.0 ¶¶¶¶¶¶¶¶¶¶¶¶.



This method needs address\_standardizer extension.

(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [state-Abbrev] [zip]

¶¶¶¶¶¶ address\_standardizer ¶¶¶¶¶¶¶¶¶¶ standardaddr ¶ norm\_addy ¶¶¶¶¶¶¶¶¶¶.  
(¶¶¶¶¶¶) ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. standardaddr ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶:

house\_num, predir, name, suftype, sufdire, unit, city, state, postcode

Enhanced: 2.4.0 norm\_addy object includes additional fields zip4 and address\_alphanumeric.

1. address ¶¶¶¶¶¶¶¶: ¶¶¶¶¶¶¶¶¶¶.
2. predirAbbrev ¶ varchar ¶¶¶¶: N, S, E, W ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. direction\_look  
¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.
3. streetName ¶ varchar ¶¶¶¶.
4. streetTypeAbbrev ¶ varchar ¶¶¶, St, Ave, Cir ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. street\_type\_lookup  
¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.
5. postdirAbbrev ¶ varchar ¶¶¶, N, S, E, W ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. direction\_lookup  
¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.
6. internal ¶ varchar ¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.
7. location ¶ varchar ¶¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.
8. stateAbbrev ¶ varchar ¶¶¶, MA, NY, MI ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶ (州名) ¶¶¶. state\_lookup  
¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.
9. zip ¶ varchar ¶¶¶¶. 02109 ¶¶¶¶¶¶¶¶¶¶¶¶¶¶.
10. parsed ¶¶ (boolean) ¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. normalize\_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.

¶¶

¶¶¶¶¶¶

```
SELECT addy.*
FROM pagc_normalize_address('9000 E R00 ST STE 999, Springfield, CO') AS addy;
```

| address     | predirabbrev | streetname | streettypeabbrev | postdirabbrev | internal  |  |
|-------------|--------------|------------|------------------|---------------|-----------|--|
| location    | stateabbrev  | zip        | parsed           |               |           |  |
| 9000        | E            | R00        | ST               |               | SUITE 999 |  |
| SPRINGFIELD | CO           |            | t                |               |           |  |

¶¶¶¶¶¶. ¶¶ postgis tiger geocoder ¶ address\_standardizer ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶  
¶¶¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶, ¶¶¶  
¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶ normaddy ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶, **Geocode** ¶¶  
¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶ **Normalize Address** ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶ address\_standardizer ¶  
standardize\_address ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

```
WITH g AS (SELECT address, ROW((sa).house_num, (sa).predir, (sa).name
, (sa).sufstype, (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               |

¶¶

**Normalize\_Address, Geocode**

12.2.15 Pprint\_Addy

Pprint\_Addy — norm\_addy ¶¶¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. ¶¶¶¶¶  
normalize\_address ¶¶¶¶¶¶¶¶¶¶¶¶.

Synopsis

varchar **pprint\_addy**(norm\_addy in\_addy);





[illegible]

| st1                             | st2                             | st3 | cross_str  |
|---------------------------------|---------------------------------|-----|------------|
| 5 Bradford St, Boston, MA 02118 | 49 Waltham St, Boston, MA 02118 |     | Waltham St |

**Geocode** 2

```
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                                         | int_addr1       | lon       | lat      | cross1        | cross2 |
|-----------------------------------------------------|-----------------|-----------|----------|---------------|--------|
| 529 Main Street, Boston MA, 02129                   |                 | -71.07181 | 42.38359 | 527 Main St,  |        |
| Boston, MA 02129                                    | Medford St      |           |          |               |        |
| 77 Massachusetts Avenue, Cambridge, MA 02139        |                 | -71.09428 | 42.35988 | 77            |        |
| Massachusetts Ave, Cambridge, MA 02139              | Vassar St       |           |          |               |        |
| 26 Capen Street, Medford, MA                        |                 | -71.12377 | 42.41101 | 9 Edison Ave, |        |
| Medford, MA 02155                                   | Capen St        |           |          | Tesla Ave     |        |
| 124 Mount Auburn St, Cambridge, Massachusetts 02138 |                 | -71.12304 | 42.37328 | 3 University  |        |
| Rd, Cambridge, MA 02138                             | Mount Auburn St |           |          |               |        |
| 950 Main Street, Worcester, MA 01610                |                 | -71.82368 | 42.24956 | 3 Maywood St, |        |
| Worcester, MA 01603                                 | Main St         |           |          | Maywood Pl    |        |

**Pprint\_Addy, Pprint\_Addy, ST\_AsText**

### 12.2.17 Topology\_Load\_Tiger

Topology\_Load\_Tiger — PostGIS TIGER

#### Synopsis

text **Topology\_Load\_Tiger**(varchar topo\_name, varchar region\_type, varchar region\_id);

PostGIS TIGER

**Note**

安裝 TIGER 數據到 PostGIS 資料庫。請參閱 Chapter 9 Section 2.2.3。安裝數據時，請確保數據庫連接字符串正確。安裝數據時，請確保數據庫連接字符串正確。

**Note**

安裝數據時，請確保數據庫連接字符串正確。安裝數據時，請確保數據庫連接字符串正確。

安裝：

1. topo\_name - 安裝到 PostGIS 資料庫。
2. region\_type - 安裝到 PostGIS 資料庫。place 和 county 數據。安裝到 tiger.place, tiger.county 數據庫。
3. region\_id - TIGER ID(geoid) 安裝到 PostGIS 資料庫。place 和 county 數據。安裝到 tiger.place 和 tiger.county 數據庫。

2.0.0 安裝數據。

安裝：安裝數據。

安裝數據 (2249) 安裝到 PostGIS 資料庫。0.25 安裝到 PostGIS 資料庫。安裝到 TIGER 數據庫。

```
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: CreateTopology

Example: CreateTopology (26986) with precision 0.25, TIGER, county, 25025.

```
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
```

Example

CreateTopology, CreateTopoGeom, TopologySummary, ValidateTopology

12.2.18 Set\_Geocode\_Setting

Set\_Geocode\_Setting — Set geocode settings.

Synopsis

text Set\_Geocode\_Setting(text setting\_name, text setting\_value);

Example

tiger.geocode\_settings returns the geocode settings. Get\_Geocode\_Setting returns the geocode settings.

2.1.0 Set\_Geocode\_Setting.

範例: 設定 Geocode

執行 `Geocode` 時, NOTICE 訊息會顯示在控制台。

```
SELECT set_geocode_setting('debug_geocode_address', 'true') As result;  
result  
-----  
true
```

範例

`Get_Geocode_Setting`

## Chapter 13

# PostGIS Special Functions Index

### 13.1 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.

- **CG\_3DUnion** - Perform 3D union using postgis\_sfcgal.
  - **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\_ClusterDBSCAN** - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
  - **ST\_ClusterIntersecting** - Aggregate function that clusters input geometries into connected sets.
  - **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\_ClusterWithin** - Aggregate function that clusters geometries by separation distance.
  - **ST\_ClusterWithinWin** - Window function that returns a cluster id for each input geometry, clustering using separation distance.
  - **ST\_Collect** - Creates a GeometryCollection or Multi\* geometry from a set of geometries.
  - **ST\_CoverageClean** - Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.
  - **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.
  - **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\_MakeLine** - `geom1, geom2, geom3, geom4, geom5, geom6, geom7, geom8, geom9, geom10, geom11, geom12, geom13, geom14, geom15, geom16, geom17, geom18, geom19, geom20, geom21, geom22, geom23, geom24, geom25, geom26, geom27, geom28, geom29, geom30, geom31, geom32, geom33, geom34, geom35, geom36, geom37, geom38, geom39, geom40, geom41, geom42, geom43, geom44, geom45, geom46, geom47, geom48, geom49, geom50, geom51, geom52, geom53, geom54, geom55, geom56, geom57, geom58, geom59, geom60, geom61, geom62, geom63, geom64, geom65, geom66, geom67, geom68, geom69, geom70, geom71, geom72, geom73, geom74, geom75, geom76, geom77, geom78, geom79, geom80, geom81, geom82, geom83, geom84, geom85, geom86, geom87, geom88, geom89, geom90, geom91, geom92, geom93, geom94, geom95, geom96, geom97, geom98, geom99, geom100`.
- **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** - `geom1, geom2, geom3, geom4, geom5, geom6, geom7, geom8, geom9, geom10, geom11, geom12, geom13, geom14, geom15, geom16, geom17, geom18, geom19, geom20, geom21, geom22, geom23, geom24, geom25, geom26, geom27, geom28, geom29, geom30, geom31, geom32, geom33, geom34, geom35, geom36, geom37, geom38, geom39, geom40, geom41, geom42, geom43, geom44, geom45, geom46, geom47, geom48, geom49, geom50, geom51, geom52, geom53, geom54, geom55, geom56, geom57, geom58, geom59, geom60, geom61, geom62, geom63, geom64, geom65, geom66, geom67, geom68, geom69, geom70, geom71, geom72, geom73, geom74, geom75, geom76, geom77, geom78, geom79, geom80, geom81, geom82, geom83, geom84, geom85, geom86, geom87, geom88, geom89, geom90, geom91, geom92, geom93, geom94, geom95, geom96, geom97, geom98, geom99, geom100`.
- **ST\_Union** - Computes a geometry representing the point-set union of the input geometries.
- **ST\_Union** - `geom1, geom2, geom3, geom4, geom5, geom6, geom7, geom8, geom9, geom10, geom11, geom12, geom13, geom14, geom15, geom16, geom17, geom18, geom19, geom20, geom21, geom22, geom23, geom24, geom25, geom26, geom27, geom28, geom29, geom30, geom31, geom32, geom33, geom34, geom35, geom36, geom37, geom38, geom39, geom40, geom41, geom42, geom43, geom44, geom45, geom46, geom47, geom48, geom49, geom50, geom51, geom52, geom53, geom54, geom55, geom56, geom57, geom58, geom59, geom60, geom61, geom62, geom63, geom64, geom65, geom66, geom67, geom68, geom69, geom70, geom71, geom72, geom73, geom74, geom75, geom76, geom77, geom78, geom79, geom80, geom81, geom82, geom83, geom84, geom85, geom86, geom87, geom88, geom89, geom90, geom91, geom92, geom93, geom94, geom95, geom96, geom97, geom98, geom99, geom100`.
- **TopoElementArray\_Agg** - Returns a `topoelementarray` for a set of `element_id`, type arrays (`topoelements`).

## 13.2 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\_CoverageClean** - Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.
- **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.

### 13.3 PostGIS SQL-MM Compliant Functions

The functions given below are PostGIS functions that conform to the SQL/MM 3 standard

- **CG\_3DArea** - 3  $\times$  8 integer array. Returns 0 if empty.
- **CG\_3DDifference** - 3  $\times$  8 integer array.
- **CG\_3DIntersection** - 3  $\times$  8 integer array.
- **CG\_3DUnion** - Perform 3D union using postgis\_sfcgal.
- **CG\_Volume** - 3  $\times$  8 integer array. Returns 0 if empty.
- **ST\_3DArea** - 3  $\times$  8 integer array. Returns 0 if empty.



- **ST\_3DDWithin** - Tests if two 3D geometries are within a given 3D distance
- **ST\_3DDifference** - 3 维差集。
- **ST\_3DDistance** - 返回两个 3D 几何体 (SRS 相同) 3 维最短距离。
- **ST\_3DIntersection** - 3 维交集。
- **ST\_3DIntersects** - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- **ST\_3DLength** - 3 维长度。
- **ST\_3DPerimeter** - 3 维周长。
- **ST\_3DUnion** - Perform 3D union.
- **ST\_AddEdgeModFace** - 添加边到面, 返回面 ID, 面 ID, 面 ID。
- **ST\_AddEdgeNewFaces** - 添加边到面, 返回面 ID, 面 ID, 面 ID。
- **ST\_AddIsoEdge** - 添加孤立边 anode 和 anothernode 之间的 alinestring 的 ID。
- **ST\_AddIsoNode** - 添加孤立节点 (isolated) 的 ID。返回 NULL 表示失败, 返回面 ID 表示成功。
- **ST\_Area** - 3 维面积。
- **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsGML** - 返回 GML 2 或 GML 3 的字符串。
- **ST\_AsText** - 返回 WKT(Well-Known Text) 字符串 SRID 信息。
- **ST\_Boundary** - 返回几何体的边界。
- **ST\_Buffer** - Computes a geometry covering all points within a given distance from a geometry.
- **ST\_Centroid** - 3 维质心。
- **ST\_ChangeEdgeGeom** - 3 维改变边几何。
- **ST\_Contains** - Tests if every point of B lies in A, and their interiors have a point in common
- **ST\_ConvexHull** - Computes the convex hull of a geometry.
- **ST\_CoordDim** - ST\_Geometry 的坐标维数。
- **ST\_CreateTopoGeo** - 3 维创建拓扑几何。
- **ST\_Crosses** - Tests if two geometries have some, but not all, interior points in common
- **ST\_CurveN** - Returns the Nth component curve geometry of a CompoundCurve.
- **ST\_CurveToLine** - Converts a geometry containing curves to a linear geometry.
- **ST\_Difference** - Computes a geometry representing the part of geometry A that does not intersect geometry B.
- **ST\_Dimension** - ST\_Geometry 的维数。

- **ST\_Disjoint** - Tests if two geometries have no points in common
- **ST\_Distance** - 返回 3 个值 (longest) 值。
- **ST\_EndPoint** - ST\_LineString 或 ST\_CircularString 的端点。
- **ST\_Envelope** - 返回 (double precision; float8) 的包络面。
- **ST\_Equals** - Tests if two geometries include the same set of points
- **ST\_ExteriorRing** - 返回多边形的外环。
- **ST\_GMLToSQL** - GML 格式化为 ST\_Geometry 格式。或 ST\_GeomFromGML 的别名。
- **ST\_GeomCollFromText** - Makes a collection Geometry from collection WKT with the given SRID. If SRID is not given, it defaults to 0.
- **ST\_GeomFromText** - WKT 格式化为 ST\_Geometry 格式。
- **ST\_GeomFromWKB** - WKB(Well-Known Binary) 格式化为 SRID 指定的 Geometry 格式。
- **ST\_GeometryFromText** - WKT(Well-Known Text) 格式化为 ST\_Geometry 格式。或 ST\_GeomFromText 的别名。
- **ST\_GeometryN** - ST\_Geometry 的 N 个几何体。
- **ST\_GeometryType** - ST\_Geometry 的几何类型。
- **ST\_GetFaceEdges** - 返回面 (face) 的边。
- **ST\_GetFaceGeometry** - 返回面 (face) 的 ID 对应的 Geometry。
- **ST\_InitTopoGeo** - Creates a new topology schema and registers it in the topology.topology table.
- **ST\_InteriorRingN** - 返回多边形 (polygon) 的 N 个内环。
- **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\_IsClosed** - LINESTRING 是否是闭合的 (TRUE 或 FALSE)。或 (LINESTRING) 是否是 TRUE 的。
- **ST\_IsEmpty** - Tests if a geometry is empty.
- **ST\_IsRing** - Tests if a LineString is closed and simple.
- **ST\_IsSimple** - 是否是简单的 (TRUE 或 FALSE)。
- **ST\_IsValid** - Tests if a geometry is well-formed in 2D.
- **ST\_Length** - 返回长度。
- **ST\_LineFromText** - 返回 SRID 指定的 WKT 格式化为 Geometry。SRID 为 0 时，返回 Geometry。
- **ST\_LineFromWKB** - 返回 SRID 指定的 WKB 格式化为 LINESTRING 格式。
- **ST\_LinestringFromWKB** - 返回 SRID 指定的 WKB 格式化为 Linestring 格式。
- **ST\_LocateAlong** - Returns the point(s) on a geometry that match a measure value.
- **ST\_LocateBetween** - Returns the portions of a geometry that match a measure range.

- **ST\_M** - Returns the M coordinate of a Point.
- **ST\_MLineFromText** - WKT `ST_MultiLineString` を読み込みます。
- **ST\_MPointFromText** - Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.
- **ST\_MPolyFromText** - Makes a MultiPolygon Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.
- **ST\_ModEdgeHeal** - Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node.
- **ST\_ModEdgeSplit** - `edge_id`, `new_node`, `new_edges` を返します。
- **ST\_MoveIsoNode** - Moves an isolated node in a topology from one point to another. If new apoint geometry exists as a node an error is thrown. Returns description of move.
- **ST\_NewEdgeHeal** - Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided.
- **ST\_NewEdgesSplit** - `edge_id`, `new_node`, `new_edges` を返します。2 つの新しい ID を返します。
- **ST\_NumCurves** - Return the number of component curves in a CompoundCurve.
- **ST\_NumGeometries** - `geometry` の数値を返します。
- **ST\_NumInteriorRings** - `geometry` の数値を返します。
- **ST\_NumPatches** - `geometry` の数値を返します。NULL を返します。
- **ST\_NumPoints** - ST\_LineString の点の数、ST\_CircularString の点の数。
- **ST\_OrderingEquals** - Tests if two geometries represent the same geometry and have points in the same directional order
- **ST\_Overlaps** - Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other
- **ST\_PatchN** - ST\_Geometry の `n` 番目のパッチを返します。
- **ST\_Perimeter** - Returns the length of the boundary of a polygonal geometry or geography.
- **ST\_Point** - Creates a Point with X, Y and SRID values.
- **ST\_PointFromText** - `SRID` `WKT` から Geometry を作成します。SRID が省略された場合は 0 です。
- **ST\_PointFromWKB** - `SRID` `WKB` から Geometry を作成します。
- **ST\_PointN** - ST\_LineString の点、ST\_CircularString の点。
- **ST\_PointOnSurface** - Computes a point guaranteed to lie in a polygon, or on a geometry.
- **ST\_Polygon** - Creates a Polygon from a LineString with a specified SRID.
- **ST\_PolygonFromText** - Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.
- **ST\_Relate** - Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix

- **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.
- **ST\_RemEdgeNewFace** - 面を削除し、新しい面を作成する。面を削除する場合は、面が空になる場合はエラーを返す。
- **ST\_RemoveIsoEdge** - Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown.
- **ST\_RemoveIsoNode** - 孤立したノードを削除する。孤立したノードとは、面が空になる場合のノード (面が空になる場合はエラーを返す) である。面が空になる場合はエラーを返す。
- **ST\_SRID** - Returns the spatial reference identifier for a geometry.
- **ST\_StartPoint** - Returns the first point of a LineString.
- **ST\_SymDifference** - Computes a geometry representing the portions of geometries A and B that do not intersect.
- **ST\_Touches** - Tests if two geometries have at least one point in common, but their interiors do not intersect
- **ST\_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST\_Union** - Computes a geometry representing the point-set union of the input geometries.
- **ST\_Volume** - 3次元ジオメトリの体積を返す。面は0を返す。
- **ST\_WKBToSQL** - WKB(Well-Known Binary) ジオメトリを ST\_Geometry オブジェクトに変換する。SRIDも指定可能。ST\_GeomFromWKB と同じ。
- **ST\_WKTToSQL** - WKT(Well-Known Text) ジオメトリを ST\_Geometry オブジェクトに変換する。ST\_GeomFromText と同じ。
- **ST\_Within** - Tests if every point of A lies in B, and their interiors have a point in common
- **ST\_X** - Returns the X coordinate of a Point.
- **ST\_Y** - Returns the Y coordinate of a Point.
- **ST\_Z** - Returns the Z coordinate of a Point.
- **ST\_SRID** - Returns the spatial reference identifier for a topogeometry.

## 13.4 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** - ジオメトリの面積を返す。

- **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsEWKT** - 返回 WKT(Well-Known Text) 格式的 SRID 信息。
- **ST\_AsGML** - 返回 GML 2 或 GML 3 格式。
- **ST\_AsGeoJSON** - Return a geometry or feature in GeoJSON format.
- **ST\_AsKML** - 返回 GML 2 或 GML 3 格式。
- **ST\_AsSVG** - Returns SVG path data for a geometry.
- **ST\_AsText** - 返回 WKT(Well-Known Text) 格式的 SRID 信息。
- **ST\_Azimuth** - 返回 2 个几何体之间的方位角。
- **ST\_Buffer** - Computes a geometry covering all points within a given distance from a geometry.
- **ST\_Centroid** - 返回几何体的质心。
- **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** - 返回 3 个几何体 (最长) 之间的距离。
- **ST\_GeogFromText** - WKT (格式) 转换为地理坐标。
- **ST\_GeogFromWKB** - WKB 格式转换为 EWKB(二进制 WKB) 格式。
- **ST\_GeographyFromText** - WKT (格式) 转换为地理坐标。
- **=** - 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** - 返回几何体的长度。
- **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** - 返回 2 个几何体之间的最短距离。
- **ST\_Summary** - 返回几何体的摘要信息。
- **<->** - A 与 B 之间的距离。
- **&&** - A 与 B 在 2D 空间中是否重合。



- **ST\_BandNoDataValue** - 返回 NODATA 值。 bandnum 1 返回。
- **ST\_BandPath** - 返回 bandnum 1 的路径。
- **ST\_BandPixelType** - 返回 bandnum 1 的像素类型。
- **ST\_Clip** - Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.
- **ST\_ColorMap** - 返回 8BUI 值 (grayscale, RGB, RGBA) 4 个值。
- **ST\_Contains** - 返回 rastA 是否包含 rastB, 返回 rastB 是否包含 rastA。
- **ST\_ContainsProperly** - rastB 是否包含 rastA。
- **ST\_Contour** - Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.
- **ST\_ConvexHull** - BandNoDataValue 返回, ST\_Envelope 返回。
- **ST\_Count** - 返回 1 的值。 exclude\_nodata\_value 返回, NODATA 返回。
- **ST\_CountAgg** - 返回 1 的值。 exclude\_nodata\_value 返回, NODATA 返回。
- **ST\_CoveredBy** - 返回 rastA 是否被 rastB 覆盖。
- **ST\_Covers** - 返回 rastB 是否覆盖 rastA。
- **ST\_DFullyWithin** - 返回 rastA 是否完全在 rastB 内部。
- **ST\_DWithin** - 返回 rastA 是否在 rastB 内部。
- **ST\_Disjoint** - 返回 rastA 是否与 rastB 不相交。
- **ST\_DumpAsPolygons** - 返回 geomval(geom, val) 返回。 bandnum 1 返回。
- **ST\_DumpValues** - 返回 2 个值。
- **ST\_Envelope** - 返回。
- **ST\_FromGDALRaster** - 返回 GDAL 返回。
- **ST\_GeoReference** - 返回 (world) 返回 GDAL 返回 ESRI 返回。
- **ST\_Grayscale** - Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue
- **ST\_HasNoBand** - 返回 1 的值。

- **ST\_Height** - 返回指定點或面的高度。
- **ST\_HillShade** - 根據高度、坡度、方位角計算 Hillshade 值。
- **ST\_Histogram** - 返回 (bin; 每個二維空間範圍內的面積) 的直方圖。
- **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** - 返回兩個二維或多維幾何體的交集。
- **ST\_IntersectionFractions** - Calculates the fraction of each raster cell that is covered by a given geometry.
- **ST\_Intersects** - 返回 rastA 是否與 rastB 相交。
- **ST\_IsEmpty** - 返回是否為空 (width = 0, height = 0) 的幾何體。
- **ST\_MakeEmptyCoverage** - Cover georeferenced area with a grid of empty raster tiles.
- **ST\_MakeEmptyRaster** - 返回一個 (width & height), 指定 X & Y, 指定, 指定 (scalex, scaley, skewx & skewy) 的幾何體 (SRID) 的幾何體。指定, 指定, 指定 SRID 的幾何體。SRID 指定, 指定 0(unknown) 的幾何體。
- **ST\_MapAlgebra (callback function version)** - 返回 1 個, 指定, 指定 1 個。
- **ST\_MapAlgebraExpr** - 返回 1 個: 指定 PostgreSQL 指定, 指定, 指定 1 個。
- **ST\_MapAlgebraExpr** - 返回 2 個: 指定 2 個 PostgreSQL 指定, 指定, 指定 1 個。
- **ST\_MapAlgebraFct** - 返回 1 個: 指定 PostgreSQL 指定, 指定, 指定 1 個。
- **ST\_MapAlgebraFct** - 返回 2 個: 指定 2 個 PostgreSQL 指定, 指定, 指定 1 個。
- **ST\_MapAlgebraFctNgb** - 返回 1 個: 指定 PostgreSQL 指定 (Map Algebra Nearest Neighbor) 指定。指定 (neighborhood) 指定 PostgreSQL 指定。
- **ST\_MapAlgebra (expression version)** - 返回 1 個 - 指定 1 個 2 個, 指定, 指定 SQL 指定 1 個。
- **ST\_MemSize** - 返回指定 (指定) 的記憶體大小。
- **ST\_MetaData** - 返回 (skew), 指定, 指定。
- **ST\_MinConvexHull** - 返回 NODATA 的幾何體。



- **ST\_NearestValue** - columnx rowy, 返回指定栅格中最近的非空值。如果指定了 NODATA 则返回 NODATA。
- **ST\_Neighborhood** - columnx rowy, 返回指定栅格中指定邻域内的所有非空值。如果指定了 NODATA 则返回 NODATA。邻域大小由 2 个参数指定。
- **ST\_NotSameAlignmentReason** - 返回指定两个栅格不同对齐的原因。
- **ST\_NumBands** - 返回指定栅格的波段数。
- **ST\_Overlaps** - 返回两个栅格是否重叠。rastA 和 rastB 是栅格名称。
- **ST\_PixelAsCentroid** - 返回指定栅格中指定像素的质心 (X, Y)。
- **ST\_PixelAsCentroids** - 返回指定栅格中指定像素的质心 (X, Y) 列表。
- **ST\_PixelAsPoint** - 返回指定栅格中指定像素的点。
- **ST\_PixelAsPoints** - 返回指定栅格中指定像素的点列表。
- **ST\_PixelAsPolygon** - 返回指定栅格中指定像素的多边形。
- **ST\_PixelAsPolygons** - 返回指定栅格中指定像素的多边形列表。
- **ST\_PixelHeight** - 返回指定栅格中指定像素的高度。
- **ST\_PixelOfValue** - 返回指定栅格中指定值的像素的 columnx, rowy。
- **ST\_PixelWidth** - 返回指定栅格中指定像素的宽度。
- **ST\_Polygon** - NODATA 返回指定栅格中指定像素的多边形。
- **ST\_Quantile** - 返回指定栅格中指定值的 (population) 分位数 (quantile) 列表。例如, 返回 25%, 50%, 75% 的分位数 (percentile) 列表。
- **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** - 返回指定栅格中指定像素的 X, Y (列, 行) 坐标。
- **ST\_RasterToWorldCoordX** - 返回指定栅格中指定像素的 X 坐标。
- **ST\_RasterToWorldCoordY** - 返回指定栅格中指定像素的 Y 坐标。
- **ST\_Reclass** - 返回指定栅格中指定像素的重新分类后的值。nband 指定重新分类的波段数。nband 指定重新分类的波段数。nband 指定重新分类的波段数。
- **ST\_ReclassExact** - Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.
- **ST\_Resample** - 返回指定栅格中指定像素的重新采样后的值。

- **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** - 调整栅格的尺寸/分辨率。
- **ST\_Reskew** - 调整 (栅格的 skew) 参数。NearestNeighbor(最近邻), Bilinear, Cubic, CubicSpline 或 Lanczos 重采样算法。最近邻 NearestNeighbor 默认。
- **ST\_Rotation** - 栅格的旋转角度。
- **ST\_Roughness** - DEM 栅格的“粗糙度” (roughness) 值。
- **ST\_SRID** - spatial\_ref\_sys 表中的 SRID 值。
- **ST\_SameAlignment** - 检查两个栅格是否具有相同的对齐方式 (是否都是上行或下行对齐)。
- **ST\_ScaleX** - 栅格的 X 轴缩放比例。
- **ST\_ScaleY** - 栅格的 Y 轴缩放比例。
- **ST\_SetBandIndex** - Update the external band number of an out-db band
- **ST\_SetBandIsNoData** - 设置 isnodata 标志。
- **ST\_SetBandNoDataValue** - NODATA 值。栅格中的 NODATA 值默认为 1。设置 NODATA 值, nodata value = NULL。
- **ST\_SetBandPath** - Update the external path and band number of an out-db band
- **ST\_SetGeoReference** - 设置栅格的地理参考信息。GDAL 的 ESRI 格式。
- **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** - 设置栅格的旋转角度。
- **ST\_SetSRID** - 设置 SRID 值。
- **ST\_SetScale** - X 和 Y 轴的缩放比例。
- **ST\_SetSkew** - 设置 X 和 Y 轴的 skew (倾斜) 参数。
- **ST\_SetUpperLeft** - Sets the value of the upper left corner of the pixel of the raster to projected X and Y coordinates.
- **ST\_SetValue** - 设置 columnx, rowy 位置的值。
- **ST\_SetValues** - 设置栅格中的值。
- **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** - X 轴的 skew (倾斜) 参数。
- **ST\_SkewY** - Y 轴的 skew (倾斜) 参数。
- **ST\_Slope** - 栅格的坡度 (slope) 值。

- **ST\_SnapToGrid** - 将栅格数据 snap 到指定的栅格。NearestNeighbor(最近邻), Bilinear, Cubic, CubicSpline 或 Lanczos 插值方法。最近邻 NearestNeighbor 是默认值。
- **ST\_Summary** - 返回栅格带的统计信息。
- **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 if 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 if no band is specified.
- **ST\_TPI** - 地形位置指数 (Topographic Position Index) 栅格函数。
- **ST\_TRI** - 地形崎岖度指数 (Terrain Ruggedness Index) 栅格函数。
- **ST\_Tile** - 将栅格数据分割成指定大小的瓦片。
- **ST\_Touches** - 栅格A 与 栅格B 是否接触, 返回 TRUE 或 FALSE。
- **ST\_Transform** - 将栅格数据从一种坐标系转换到另一种坐标系。NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos 插值方法。最近邻 NearestNeighbor 是默认值。
- **ST\_Union** - 将两个栅格合并为一个栅格。
- **ST\_UpperLeftX** - 返回栅格左上角的 X 坐标。
- **ST\_UpperLeftY** - 返回栅格左上角的 Y 坐标。
- **ST\_Value** - 返回栅格中指定列和行的值。如果指定了 exclude\_nodata\_value 为 TRUE, 则返回 nodata 值。如果指定了 exclude\_nodata\_value 为 FALSE, 则返回 nodata 值。
- **ST\_ValueCount** - 返回栅格中每个值的计数 (频率)。返回一个表, 包含值和计数。如果指定了 NODATA 为 TRUE, 则返回 nodata 值。如果指定了 NODATA 为 FALSE, 则返回 nodata 值。
- **ST\_Width** - 返回栅格的宽度。
- **ST\_Within** - 栅格B 是否完全在栅格A 内部, 返回 TRUE 或 FALSE。
- **ST\_WorldToRasterCoord** - 将世界坐标 X, Y(经纬度) 转换为栅格坐标。
- **ST\_WorldToRasterCoordX** - 将世界坐标 (pt) 转换为栅格坐标 X, Y (xw, yw)。
- **ST\_WorldToRasterCoordY** - 将世界坐标 (pt) 转换为栅格坐标 X, Y (xw, yw)。
- **UpdateRasterSRID** - 更新栅格的 SRID 值。

## 13.6 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** - 將 geomval(geom, val) 轉換為 1 個 Polygon。
- **ST\_Intersection** - 返回兩個 geometry\_dump 物件的交集。
- **ST\_Dump** - Returns a set of geometry\_dump rows for the components of a geometry.
- **ST\_DumpPoints** - 將 geometry\_dump 物件轉換為 Point 物件。
- **ST\_DumpRings** - Returns a set of geometry\_dump rows for the exterior and interior rings of a Polygon.
- **ST\_DumpSegments** - 將 geometry\_dump 物件轉換為 LineString 物件。

## 13.7 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.
- **MakeTopologyPrecise** - Snap topology vertices to precision grid.
- **Box3D** - 將 geometry\_dump 物件轉換為 BOX3D 物件。
- **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** - 將 geometry\_dump 物件轉換為 TWKB(Tiny Well-Known Binary) 物件。
- **ST\_Box2dFromGeoHash** - GeoHash 轉換為 BOX2D 物件。
- **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\_RemoveIrrelevantPointsForView** - Removes points that are irrelevant for rendering a specific rectangular view of a geometry.
- **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 Topo-Geometry objects.
- **ValidateTopology** - Returns a set of validate topology\_returntype objects detailing issues with topology.
- **ValidateTopologyPrecision** - Returns non-precise vertices in the topology.
- **~(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).
- **@(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- **@(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- **@(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- **&&(box2df,box2df)** - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- **&&(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- **&&(geometry,box2df)** - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

## 13.8 PostGIS Functions that support 3D

The functions given below are PostGIS functions that do not throw away the Z-Index.

- **AddGeometryColumn** - Adds a geometry column to a table.
- **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
- **CG\_3DAlphaWrapping** - Computes a 3D Alpha-wrapping strictly enclosing a geometry.
- **CG\_3DArea** - 3D area of a geometry. Returns 0 for non-3D geometries.
- **CG\_3DConvexHull** - Computes the 3D convex hull of a geometry.
- **CG\_3DDifference** - 3D difference of two geometries.
- **CG\_3DIntersection** - 3D intersection of two geometries.
- **CG\_3DRotate** - Rotates a geometry in 3D space around an axis vector.
- **CG\_3DScale** - Scales a geometry by separate factors along X, Y, and Z axes.

- **CG\_3DScaleAroundCenter** - Scales a geometry in 3D space around a specified center point.
- **CG\_3DTranslate** - Translates (moves) a geometry by given offsets in 3D space.
- **CG\_3DUnion** - Perform 3D union using postgis\_sfcgal.
- **CG\_ApproximateMedialAxis** - 計算幾何體的近似中軸線。
- **CG\_ConstrainedDelaunayTriangles** - Return a constrained Delaunay triangulation around the given input geometry.
- **CG\_Extrude** - 將幾何體沿 Z 軸拉伸。
- **CG\_ForceLHR** - LHR(Left Hand Reverse; 左手逆時針) 旋轉幾何體。
- **CG\_IsPlanar** - 判斷幾何體是否為平面。
- **CG\_IsSolid** - 判斷幾何體是否為實體。實體必須是封閉的、無孔的、且無自交的。
- **CG\_MakeSolid** - 將幾何體轉換為實體。如果輸入是 TIN，則返回 TIN。
- **CG\_Orientation** - 返回幾何體的 (orientation) 值。
- **CG\_RotateX** - Rotates a geometry around the X-axis by a given angle.
- **CG\_RotateY** - Rotates a geometry around the Y-axis by a given angle.
- **CG\_RotateZ** - Rotates a geometry around the Z-axis by a given angle.
- **CG\_Simplify** - Reduces the complexity of a geometry while preserving essential features and Z/M values.
- **CG\_StraightSkeleton** - 計算幾何體的 (straight skeleton) 值。
- **CG\_Tessellate** - 將幾何體轉換為 TIN (tessellation) 值。TIN 值可以為 TIN 或 TIN 的集合。
- **CG\_Visibility** - Compute a visibility polygon from a point or a segment in a polygon geometry
- **CG\_Volume** - 3D 幾何體的體積。返回幾何體的 (volume) 0 值。
- **DropGeometryColumn** - 刪除幾何體欄位。
- **GeometryType** - ST\_Geometry 幾何體的類型。
- **ST\_3DArea** - 3D 幾何體的面积。返回幾何體的 0 值。
- **ST\_3DClosestPoint** - g2 幾何體到 g1 幾何體的 3D 最近點。返回幾何體的 3D 最近點。
- **ST\_3DConvexHull** - 幾何體的 3D 凸包。
- **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** - 3D 幾何體的差集。
- **ST\_3DDistance** - 幾何體到幾何體 (SRS 幾何體) 3D 距離。返回幾何體的 3D 距離。
- **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST\_3DIntersection** - 3D 幾何體的交集。
- **ST\_3DIntersects** - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)

- **ST\_3DLength** - 返回3D几何体的长度。
- **ST\_3DLineInterpolatePoint** - Returns a point interpolated along a 3D line at a fractional location.
- **ST\_3DLongestLine** - 返回3D几何体中3条最长的线。
- **ST\_3DMaxDistance** - 返回3D几何体中, 最大距离 (SRS 坐标系) 3 个点之间的最大距离。
- **ST\_3DPerimeter** - 返回3D几何体的周长。
- **ST\_3DShortestLine** - 返回3D几何体中3条最短的线。
- **ST\_3DUnion** - Perform 3D union.
- **ST\_AddMeasure** - Interpolates measures along a linear geometry.
- **ST\_AddPoint** - 向3D几何体中添加点。
- **ST\_Affine** - Apply a 3D affine transformation to a geometry.
- **ST\_ApproximateMedialAxis** - 返回3D几何体的近似中轴。
- **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** - 返回 WKT(Well-Known Text) 格式的 SRID 几何体。
- **ST\_AsGML** - 返回 GML 2 或 GML 3 格式的几何体。
- **ST\_AsGeoJSON** - Return a geometry or feature in GeoJSON format.
- **ST\_AsHEXEWKB** - 返回 (NDR) 或 (XDR) 格式的 HEXEWKB (二进制) 几何体。
- **ST\_AsKML** - 返回 GML 2 或 GML 3 格式的几何体。
- **ST\_AsX3D** - 返回 X3D XML 格式的几何体: ISO-IEC-19776-1.2-X3DEncodings-XML 格式。
- **ST\_Boundary** - 返回3D几何体的边界。
- **ST\_BoundingDiagonal** - 返回3D几何体的边界对角线。
- **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** - ST\_Geometry 的坐标维度。
- **ST\_CurveN** - Returns the Nth component curve geometry of a CompoundCurve.
- **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** - 返回几何体中所有点的集合。
- **ST\_DumpRings** - Returns a set of geometry\_dump rows for the exterior and interior rings of a Polygon.
- **ST\_DumpSegments** - 返回几何体中所有线段的集合。
- **ST\_EndPoint** - ST\_LineString 或 ST\_CircularString 返回几何体的终点。
- **ST\_ExteriorRing** - 返回多边形的外环。
- **ST\_Extrude** - 将二维几何体沿 Z 轴拉伸。
- **ST\_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
- **ST\_Force2D** - 强制将几何体转换为 2D。
- **ST\_ForceCurve** - 强制将几何体转换为曲线 (upcast)。
- **ST\_ForceLHR** - LHR(Left Hand Reverse; 左手规则) 强制将几何体转换为左手规则。
- **ST\_ForcePolygonCCW** - Orients all exterior rings counter-clockwise and all interior rings clockwise.
- **ST\_ForcePolygonCW** - Orients all exterior rings clockwise and all interior rings counter-clockwise.
- **ST\_ForceRHR** - 强制将几何体转换为右手规则 (orientation) (Right-Hand Rule)。
- **ST\_ForceSFS** - 强制将几何体转换为 SFS 1.1 格式。
- **ST\_Force3D** - 强制将几何体转换为 3D。ST\_Force3DZ 强制将几何体转换为 3DZ。
- **ST\_Force3DZ** - 强制将几何体转换为 3DZ。
- **ST\_Force4D** - 强制将几何体转换为 4D。
- **ST\_ForceCollection** - 强制将几何体转换为集合。
- **ST\_GeomFromEWKB** - EWKB(Extended Well-Known Binary) 返回 ST\_Geometry。
- **ST\_GeomFromEWKT** - EWKT(Extended Well-Known Text) 返回 ST\_Geometry。
- **ST\_GeomFromGML** - 从 GML 返回 PostGIS 几何体。
- **ST\_GeomFromGeoJSON** - 从 GeoJSON 返回 PostGIS 几何体。
- **ST\_GeomFromKML** - 从 KML 返回 PostGIS 几何体。
- **ST\_GeometricMedian** - 返回几何体的几何中位数 (median)。
- **ST\_GeometryN** - ST\_Geometry 返回几何体的第 N 个分量。
- **ST\_GeometryType** - ST\_Geometry 返回几何体的类型。
- **ST\_HasArc** - Tests if a geometry contains a circular arc
- **ST\_HasM** - Checks if a geometry has an M (measure) dimension.
- **ST\_HasZ** - Checks if a geometry has a Z dimension.



- **ST\_InteriorRingN** - 返回几何体的第 N 个内部环。
- **ST\_InterpolatePoint** - 返回沿几何体 (M 值) 的指定比例位置的点。
- **ST\_Intersection** - Computes a geometry representing the shared portion of geometries A and B.
- **ST\_IsClosed** - LINESTRING 是否是闭合的 TRUE 或 FALSE。MULTILINESTRING (MULTILINESTRING) 是否是 TRUE 或 FALSE。
- **ST\_IsCollection** - 是否是集合, 点, 面或线 TRUE 或 FALSE。
- **ST\_IsPlanar** - 是否是平面。
- **ST\_IsPolygonCCW** - Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.
- **ST\_IsPolygonCW** - Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.
- **ST\_IsSimple** - 是否是简单的 TRUE 或 FALSE。
- **ST\_IsSolid** - 是否是实体的。MULTISURFACE 是否是。
- **ST\_IsValidTrajectory** - Tests if the geometry is a valid trajectory.
- **ST\_LengthSpheroid** - 返回球面长度。
- **ST\_LineFromMultiPoint** - 从多点创建线。
- **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** - 从点, 面或线创建线。
- **ST\_MakePoint** - Creates a 2D, 3DZ or 4D Point.
- **ST\_MakePolygon** - Creates a Polygon from a shell and optional list of holes.
- **ST\_MakeSolid** - 从面创建实体。从面创建实体。从面创建实体, 从面创建实体 TIN 面。
- **ST\_MakeValid** - Attempts to make an invalid geometry valid without losing vertices.
- **ST\_MemSize** - ST\_Geometry 的内存大小。
- **ST\_MemUnion** - Aggregate function which unions geometries in a memory-efficient but slower way
- **ST\_NDims** - ST\_Geometry 的维度。
- **ST\_NPoints** - 返回几何体 (面) 的点数。
- **ST\_NRings** - 返回几何体 (面) 的环数。
- **ST\_Node** - Nodes a collection of lines.
- **ST\_NumCurves** - Return the number of component curves in a CompoundCurve.
- **ST\_NumGeometries** - 返回几何体集合中的几何体数量。MULTISURFACE 集合。

- **ST\_NumPatches** - 返回几何体中的补丁数量。如果几何体为 NULL，则返回 NULL。
- **ST\_Orientation** - 返回几何体的方位 (orientation)。
- **ST\_PatchN** - 返回几何体中的第 N 个补丁。
- **ST\_PointFromWKB** - 从 WKB 数据中创建点，指定 SRID。
- **ST\_PointN** - 返回几何体中的第 N 个点。
- **ST\_PointOnSurface** - 计算一个点，保证位于多边形或几何体内。
- **ST\_Points** - 返回几何体中的所有点。
- **ST\_Polygon** - 从 LineString 创建多边形，指定 SRID。
- **ST\_RemovePoint** - 从线串中移除一个点。
- **ST\_RemoveRepeatedPoints** - 返回移除重复点后的几何体版本。
- **ST\_Reverse** - 返回几何体的反向。
- **ST\_Rotate** - 绕原点旋转几何体。
- **ST\_RotateX** - 绕 X 轴旋转几何体。
- **ST\_RotateY** - 绕 Y 轴旋转几何体。
- **ST\_RotateZ** - 绕 Z 轴旋转几何体。
- **ST\_Scale** - 按给定因子缩放几何体。
- **ST\_Scroll** - 更改封闭 LineString 的起始点。
- **ST\_SetPoint** - 将几何体中的点替换为指定点。
- **ST\_ShiftLongitude** - 将几何体的经度坐标在 -180..180 和 0..360 之间归一化。
- **ST\_SnapToGrid** - 将几何体中的点 snap 到指定的栅格。
- **ST\_StartPoint** - 返回线串的起始点。
- **ST\_StraightSkeleton** - 返回几何体的直线骨架。
- **ST\_SwapOrdinates** - 交换几何体中点的纵坐标。
- **ST\_SymDifference** - 计算两个几何体 A 和 B 的对称差集。
- **ST\_Tessellate** - 将几何体 tessellate 为三角形 (TIN)。
- **ST\_TransScale** - 按给定偏移和因子缩放并平移几何体。
- **ST\_Translate** - 按给定偏移平移几何体。
- **ST\_UnaryUnion** - 计算单个几何体所有组件的并集。
- **ST\_Union** - 计算多个几何体的并集。
- **ST\_Volume** - 计算 3D 几何体的体积。
- **ST\_WrapX** - 将 X 坐标归一化到指定范围。
- **ST\_X** - 返回点的 X 坐标。
- **ST\_XMax** - 返回 2D 或 3D 包围盒的 X 最大值。

- **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** - ST\_Geometry flag.
- **Equals** - TopoGeometry flag.
- **Intersects** - TopoGeometry flag.
- **UpdateGeometrySRID** - Updates the SRID of all features in a geometry column, and the table meta-data.
- **&&&** - A n-D bounding box B n-D bounding box TRUE flag.
- **&&&(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.

## 13.9 PostGIS Curved Geometry Support Functions

The functions given below are PostGIS functions that can use CIRCULARSTRING, CURVEPOLYGON, and other curved geometry types

- **AddGeometryColumn** - flag.
- **Box2D** - Returns a BOX2D representing the 2D extent of a geometry.
- **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
- **DropGeometryColumn** - flag.
- **GeometryType** - ST\_Geometry flag.
- **PostGIS\_AddBBox** - flag.
- **PostGIS\_DropBBox** - flag.
- **PostGIS\_HasBBox** - flag, flag.
- **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** - 将 WKT(Well-Known Text) 转换为 SRID 指定的几何类型。
- **ST\_AsHEXEWKB** - 将 (NDR) 或 (XDR) 格式的 HEXEWKB (二进制) 转换为二进制格式。
- **ST\_AsSVG** - Returns SVG path data for a geometry.
- **ST\_AsText** - 将/将 WKT(Well-Known Text) 转换为 SRID 指定的几何类型。
- **ST\_ClusterDBSCAN** - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- **ST\_ClusterWithin** - Aggregate function that clusters geometries by separation distance.
- **ST\_ClusterWithinWin** - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- **ST\_Collect** - Creates a GeometryCollection or Multi\* geometry from a set of geometries.
- **ST\_CoordDim** - ST\_Geometry 的坐标维度。
- **ST\_CurveToLine** - Converts a geometry containing curves to a linear geometry.
- **ST\_Distance** - 返回 3 个 (最长) 距离。
- **ST\_Dump** - Returns a set of geometry\_dump rows for the components of a geometry.
- **ST\_DumpPoints** - 将几何体分解为点。
- **ST\_EndPoint** - ST\_LineString 或 ST\_CircularString 的终点。
- **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** - 将 "2 维" 强制为 2D。
- **ST\_ForceCurve** - 将几何体强制为曲线 (upcast)。
- **ST\_ForceSFS** - 将 SFS 1.1 强制为 SFS 1.1。
- **ST\_Force3D** - 将 XYZ 强制为 3D。ST\_Force3DZ 强制为 3DZ。
- **ST\_Force3DM** - 将 XYM 强制为 3DM。
- **ST\_Force3DZ** - 将 XYZ 强制为 3DZ。
- **ST\_Force4D** - 将 XYZM 强制为 4D。
- **ST\_ForceCollection** - 将几何体强制为集合。
- **ST\_GeoHash** - 将几何体转换为 GeoHash。
- **ST\_GeogFromWKB** - WKB 转换为 EWKB(或 WKB) 的地理几何。
- **ST\_GeomFromEWKB** - EWKB(Extended Well-Known Binary) 转换为 ST\_Geometry 几何。
- **ST\_GeomFromEWKT** - EWKT(Extended Well-Known Text) 转换为 ST\_Geometry 几何。
- **ST\_GeomFromText** - WKT 转换为 ST\_Geometry 几何。
- **ST\_GeomFromWKB** - WKB(Well-Known Binary) 转换为 SRID 指定的几何类型。
- **ST\_GeometryN** - ST\_Geometry 的第 N 个几何体。

- **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **&<|** - A 几何对象 B 几何对象 返回 TRUE 或 FALSE。
- **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** - LINESTRING 几何对象 返回 TRUE 或 FALSE。 几何对象 (几何对象) 返回 TRUE 或 FALSE。
- **ST\_IsCollection** - 几何对象, 几何, 几何对象 返回 TRUE 或 FALSE。
- **ST\_IsEmpty** - Tests if a geometry is empty.
- **ST\_LineToCurve** - Converts a linear geometry to a curved geometry.
- **ST\_MemSize** - ST\_Geometry 几何对象。
- **ST\_NPoints** - 几何对象 (几何) 几何对象。
- **ST\_NRings** - 几何对象 几何对象。
- **ST\_PointFromWKB** - 几何 SRID 几何 WKB 几何对象。
- **ST\_PointN** - ST\_LineString 几何 ST\_CircularString 几何对象。
- **ST\_Points** - 几何对象 几何对象。
- **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** - 几何对象 几何对象。
- **ST\_SwapOrdinates** - 几何对象 几何对象。
- **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** - ST\_Geometry 几何对象。

- **UpdateGeometrySRID** - Updates the SRID of all features in a geometry column, and the table meta-data.
- **~(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).
- **&&** - A 国 2D 国国国国 B 国 2D 国国国国国国国国国国 TRUE 国国国国国国.
- **&&&** - A 国 n 国国国国国国 B 国 n 国国国国国国国国国国国国 TRUE 国国国国国国.
- **@(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.

## 13.10 PostGIS Polyhedral Surface Support Functions

The functions given below are PostGIS functions that can use POLYHEDRALSURFACE, POLYHEDRAL-SURFACEM geometries

- **Box2D** - Returns a BOX2D representing the 2D extent of a geometry.
- **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
- **CG\_3DArea** - 3 国国国国国国国国国国国国. 国国国国 0 国国国国国国国国.
- **CG\_3DConvexHull** - 国国国国国国国国国国国国国国.
- **CG\_3DDifference** - 3 国国国国国国国国国国.
- **CG\_3DIntersection** - 3 国国国国国国国国国国.
- **CG\_3DUnion** - Perform 3D union using postgis\_sfcal.

- **CG\_ApproximateMedialAxis** - 计算多边形的 medial axis.
- **CG\_Extrude** - 将 2D 几何体沿指定方向拉伸。
- **CG\_ForceLHR** - LHR(Left Hand Reverse; 左手规则) 强制使用。
- **CG\_IsPlanar** - 检查几何体是否共面。
- **CG\_IsSolid** - 检查几何体是否为实心。返回 true 表示是实心，false 表示不是。
- **CG\_MakeSolid** - 将 2D 几何体转换为 3D 实心体。返回 true 表示成功，false 表示失败。
- **CG\_StraightSkeleton** - 计算直骨架 (straight skeleton)。
- **CG\_Tessellate** - 将 TIN (Triangulated Irregular Network) 转换为 TIN 格式。
- **CG\_Visibility** - Compute a visibility polygon from a point or a segment in a polygon geometry
- **CG\_Volume** - 3 维几何体的体积。返回 0 表示无效。
- **GeometryType** - ST\_Geometry 类型的名称。
- **ST\_3DArea** - 3 维几何体的面积。返回 0 表示无效。
- **ST\_3DClosestPoint** - g2 中的点 g1 中的 3 维最近点。返回 3D 点。
- **ST\_3DConvexHull** - 计算 3D 凸包。
- **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** - 3 维几何体的差集。
- **ST\_3DDistance** - 计算 3D 距离 (SRS 无关) 3 维几何体之间的最小距离。
- **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST\_3DIntersection** - 3 维几何体的交集。
- **ST\_3DIntersects** - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- **ST\_3DLongestLine** - 计算 3 维几何体 (longest) 的最长线。
- **ST\_3DMaxDistance** - 计算 3D 最大距离 (SRS 无关) 3 维几何体之间的最大距离。
- **ST\_3DShortestLine** - 计算 3 维几何体 (shortest) 的最短线。
- **ST\_3DUnion** - Perform 3D union.
- **ST\_Affine** - Apply a 3D affine transformation to a geometry.
- **ST\_ApproximateMedialAxis** - 计算多边形的 medial axis。
- **ST\_Area** - 计算 2D 面积。
- **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** - 将 WKT(Well-Known Text) 转换为 SRID 指定的数据库格式。
- **ST\_AsGML** - 将 GML 2 或 GML 3 转换为数据库格式。
- **ST\_AsX3D** - 将 X3D XML 转换为数据库格式: ISO-IEC-19776-1.2-X3DEncodings-XML 格式。
- **ST\_CoordDim** - ST\_Geometry 的坐标维度。
- **ST\_Dimension** - ST\_Geometry 的维度。
- **ST\_Dump** - Returns a set of geometry\_dump rows for the components of a geometry.
- **ST\_DumpPoints** - 将点几何体转换为点集合。
- **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\_Extrude** - 将几何体沿 Z 轴拉伸。
- **ST\_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
- **ST\_Force2D** - 将几何体强制转换为 2D。
- **ST\_ForceLHR** - LHR(Left Hand Reverse; 左手规则) 强制几何体。
- **ST\_ForceRHR** - 强制几何体 (orientation) 符合右手规则 (Right-Hand Rule)。
- **ST\_ForceSFS** - 将 SFS 1.1 强制几何体。
- **ST\_Force3D** - 将 XYZ 强制几何体。ST\_Force3DZ 强制几何体。
- **ST\_Force3DZ** - 将 XYZ 强制几何体。
- **ST\_ForceCollection** - 将几何体强制转换为集合。
- **ST\_GeomFromEWKB** - EWKB(Extended Well-Known Binary) 转换为 ST\_Geometry 格式。
- **ST\_GeomFromEWKT** - EWKT(Extended Well-Known Text) 转换为 ST\_Geometry 格式。
- **ST\_GeomFromGML** - 将 GML 转换为 PostGIS 数据库格式。
- **ST\_GeometryN** - ST\_Geometry 的 N 个几何体。
- **ST\_GeometryType** - ST\_Geometry 的几何类型。
- **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **&<|** - A 与 B 的交集是否为空。TRUE 表示为空。
- **~=** - A 与 B 的差集是否为空。TRUE 表示为空。
- **ST\_IsClosed** - LINESTRING 是否为闭合。TRUE 表示是。POINT 类型 (POINT) 强制 TRUE。
- **ST\_IsPlanar** - 几何体是否为平面。
- **ST\_IsSolid** - 几何体是否为实心。强制几何体。
- **ST\_MakeSolid** - 强制几何体为实心。强制几何体。强制几何体, 强制几何体 TIN 强制几何体。
- **ST\_MemSize** - ST\_Geometry 的内存大小。
- **ST\_NPoints** - 几何体 (POINT) 的点数。



- **ST\_NumGeometries** - 返回几何对象中的几何数量。返回几何对象中的几何数量。
- **ST\_NumPatches** - 返回几何对象中的几何数量。返回几何对象中的几何数量 NULL 表示没有几何对象。
- **ST\_PatchN** - ST\_Geometry 返回几何对象中的几何数量。
- **ST\_RemoveRepeatedPoints** - Returns a version of a geometry with duplicate points removed.
- **ST\_Reverse** - 返回几何对象的反向几何对象。
- **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\_ShiftLongitude** - Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- **ST\_StraightSkeleton** - 返回几何对象的直线骨架 (straight skeleton) 几何对象。
- **ST\_Summary** - 返回几何对象的摘要信息。
- **ST\_SwapOrdinates** - 返回几何对象的坐标轴互换后的几何对象。
- **ST\_Tessellate** - 返回几何对象的三角化 (tessellation) 结果。返回几何对象的 TIN 或 TIN 集合。
- **ST\_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST\_Volume** - 3 维几何对象的体积。返回几何对象的体积 (或面积) 0 表示没有几何对象。
- **~(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).
- **&&** - A 2D 几何对象 B 2D 几何对象 TRUE 表示相交。
- **&&&** - A n 维几何对象 B n 维几何对象 TRUE 表示相交。
- **@(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.

## 13.11 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.
- 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 |
|-----------------------|------|------|------|--------|--------|----|---|
| ST_Collect            | ✓    |      | ✓    | ✓      |        |    |   |
| ST_LineFromMultiPoint | ✓    |      | ✓    |        |        |    |   |
| ST_MakeEnvelope       | ✓    |      |      |        |        |    |   |
| ST_MakeLine           | ✓    |      | ✓    |        |        |    |   |
| ST_MakePoint          | ✓    |      | ✓    |        |        |    |   |
| ST_MakePointM         | ✓    |      |      |        |        |    |   |
| ST_MakePolygon        | ✓    |      | ✓    |        |        |    |   |
| ST_Point              | ✓    |      |      |        | ✓      |    |   |
| ST_PointZ             | ✓    |      |      |        |        |    |   |
| ST_PointM             | ✓    |      |      |        |        |    |   |
| ST_PointZM            | ✓    |      |      |        |        |    |   |

| Function            | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|---------------------|------|------|------|--------|--------|----|---|
| ST_Polygon          | ✓    |      | ✓    |        | ✓      |    |   |
| ST_TileEnvelope     | ✓    |      |      |        |        |    |   |
| ST_HexagonGrid      | ✓    |      |      |        |        |    |   |
| ST_Hexagon          | ✓    |      |      |        |        |    |   |
| ST_SquareGrid       | ✓    |      |      |        |        |    |   |
| ST_Square           | ✓    |      |      |        |        |    |   |
| ST_Letters          | ✓    |      |      |        |        |    |   |
| GeometryType        | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_Boundary         | ✓    |      | ✓    |        | ✓      |    |   |
| ST_BoundingDiagonal | ✓    |      | ✓    |        |        |    |   |
| ST_CoordDim         | ✓    |      | ✓    | ✓      | ✓      | ✓  | ✓ |
| ST_Dimension        | ✓    |      |      |        | ✓      | ✓  | ✓ |
| ST_Dump             | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_DumpPoints       | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_DumpSegments     | ✓    |      | ✓    |        |        |    | ✓ |
| ST_DumpRings        | ✓    |      | ✓    |        |        |    |   |
| ST_EndPoint         | ✓    |      | ✓    | ✓      | ✓      |    |   |
| ST_Envelope         | ✓    |      |      |        | ✓      |    |   |
| ST_ExteriorRing     | ✓    |      | ✓    |        | ✓      |    |   |
| ST_GeometryN        | ✓    |      | ✓    | ✓      | ✓      | ✓  | ✓ |
| ST_GeometryType     | ✓    |      | ✓    |        | ✓      | ✓  |   |
| ST_HasArc           | ✓    |      | ✓    | ✓      |        |    |   |
| ST_InteriorRingN    | ✓    |      | ✓    |        | ✓      |    |   |
| ST_NumCurves        | ✓    |      | ✓    |        | ✓      |    |   |
| ST_CurveN           | ✓    |      | ✓    |        | ✓      |    |   |
| ST_IsClosed         | ✓    |      | ✓    | ✓      | ✓      | ✓  |   |
| ST_IsCollection     | ✓    |      | ✓    | ✓      |        |    |   |
| ST_IsEmpty          | ✓    |      |      | ✓      | ✓      |    |   |
| ST_IsPolygonCCW     | ✓    |      | ✓    |        |        |    |   |
| ST_IsPolygonCW      | ✓    |      | ✓    |        |        |    |   |
| ST_IsRing           | ✓    |      |      |        | ✓      |    |   |
| ST_IsSimple         | ✓    |      | ✓    |        | ✓      |    |   |
| ST_M                | ✓    |      | ✓    |        | ✓      |    |   |
| ST_MemSize          | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |

| Function                | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|-------------------------|------|------|------|--------|--------|----|---|
| ST_NDims                | ✓    |      | ✓    |        |        |    |   |
| ST_NPoints              | ✓    |      | ✓    | ✓      |        | ✓  |   |
| ST_NRings               | ✓    |      | ✓    | ✓      |        |    |   |
| ST_NumGeometries        | ✓    |      | ✓    |        | ✓      | ✓  | ✓ |
| ST_NumInteriorRings     | ✓    |      |      |        | ✓      |    |   |
| ST_NumInteriorRing      | ✓    |      |      |        |        |    |   |
| ST_NumPatches           | ✓    |      | ✓    |        | ✓      | ✓  |   |
| ST_NumPoints            | ✓    |      |      |        | ✓      |    |   |
| ST_PatchN               | ✓    |      | ✓    |        | ✓      | ✓  |   |
| ST_PointN               | ✓    |      | ✓    | ✓      | ✓      |    |   |
| ST_Points               | ✓    |      | ✓    | ✓      |        |    |   |
| ST_StartPoint           | ✓    |      | ✓    | ✓      | ✓      |    |   |
| ST_Summary              | ✓    | ✓    |      | ✓      |        | ✓  | ✓ |
| ST_X                    | ✓    |      | ✓    |        | ✓      |    |   |
| ST_Y                    | ✓    |      | ✓    |        | ✓      |    |   |
| ST_Z                    | ✓    |      | ✓    |        | ✓      |    |   |
| ST_Zmflag               | ✓    |      | ✓    | ✓      |        |    |   |
| ST_HasZ                 | ✓    |      | ✓    |        |        |    |   |
| ST_HasM                 | ✓    |      | ✓    |        |        |    |   |
| ST_AddPoint             | ✓    |      | ✓    |        |        |    |   |
| ST_CollectionExtract    | ✓    |      |      |        |        |    |   |
| ST_CollectionHomogenize | ✓    |      |      |        |        |    |   |
| ST_CurveToLine          | ✓    |      | ✓    | ✓      | ✓      |    |   |
| ST_Scroll               | ✓    |      | ✓    |        |        |    |   |
| ST_FlipCoordinates      | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_Force2D              | ✓    |      | ✓    | ✓      |        | ✓  |   |
| ST_Force3D              | ✓    |      | ✓    | ✓      |        | ✓  |   |
| ST_Force3DZ             | ✓    |      | ✓    | ✓      |        | ✓  |   |
| ST_Force3DM             | ✓    |      |      | ✓      |        |    |   |
| ST_Force4D              | ✓    |      | ✓    | ✓      |        |    |   |
| ST_ForceCollection      | ✓    |      | ✓    | ✓      |        | ✓  |   |
| ST_ForceCurve           | ✓    |      | ✓    | ✓      |        |    |   |
| ST_ForcePolygonCW       | ✓    |      | ✓    |        |        |    |   |
| ST_ForcePolygonCW       | ✓    |      | ✓    |        |        |    |   |

| Function                         | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|----------------------------------|------|------|------|--------|--------|----|---|
| ST_ForceSFS                      | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_ForceRHR                      | ✓    |      | ✓    |        |        | ✓  |   |
| ST_LineExtend                    | ✓    |      |      |        |        |    |   |
| ST_LineToCurve                   | ✓    |      | ✓    | ✓      |        |    |   |
| ST_Multi                         | ✓    |      |      |        |        |    |   |
| ST_Normalize                     | ✓    |      |      |        |        |    |   |
| ST_Project                       | ✓    | ✓    |      |        |        |    |   |
| ST_QuantizeCoordinates           | ✓    |      |      |        |        |    |   |
| ST_RemovePoint                   | ✓    |      | ✓    |        |        |    |   |
| ST_RemoveRepeatedPoints          | ✓    |      | ✓    |        |        | ✓  |   |
| ST_RemoveIrrelevantPointsForView | ✓    |      |      |        |        |    |   |
| ST_RemoveSmallParts              | ✓    |      |      |        |        |    |   |
| ST_Reverse                       | ✓    |      | ✓    |        |        | ✓  |   |
| ST_Segmentize                    | ✓    | ✓    |      |        |        |    |   |
| ST_SetPoint                      | ✓    |      | ✓    |        |        |    |   |
| ST_ShiftLongitude                | ✓    |      | ✓    |        |        | ✓  | ✓ |
| ST_WrapX                         | ✓    |      | ✓    |        |        |    |   |
| ST_SnapToGrid                    | ✓    |      | ✓    |        |        |    |   |
| ST_Snap                          | ✓    |      |      |        |        |    |   |
| ST_SwapOrdinate                  | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_IsValid                       | ✓    |      |      |        | ✓      |    |   |
| ST_IsValidDetail                 | ✓    |      |      |        |        |    |   |
| ST_IsValidReason                 | ✓    |      |      |        |        |    |   |
| ST_MakeValid                     | ✓    |      | ✓    |        |        |    |   |
| ST_InverseTransformPipeline      | ✓    |      |      |        |        |    |   |
| ST_SetSRID                       | ✓    |      |      | ✓      |        |    |   |
| ST_SRID                          | ✓    |      |      | ✓      | ✓      |    |   |
| ST_Transform                     | ✓    |      |      | ✓      | ✓      | ✓  |   |
| ST_TransformPipeline             | ✓    |      |      |        |        |    |   |
| postgis_srs_codes                |      |      |      |        |        |    |   |
| postgis_srs                      |      |      |      |        |        |    |   |
| postgis_srs_all                  |      |      |      |        |        |    |   |
| postgis_srs_search               | ✓    |      |      |        |        |    |   |
| ST_BdPolyFromText                | ✓    |      |      |        |        |    |   |
| ST_BdMPolyFromText               | ✓    |      |      |        |        |    |   |

| Function                 | geom           | geog | 2.5D | Curves | SQL MM | PS | T |
|--------------------------|----------------|------|------|--------|--------|----|---|
| ST_GeogFromText          |                | ✓    |      |        |        |    |   |
| ST_GeographyFromText     |                | ✓    |      |        |        |    |   |
| ST_GeomCollFr            | ✓ Text         |      |      |        | ✓      |    |   |
| ST_GeomFromE             | ✓ CT           |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_GeomFromM             | ✓ RC21         |      |      |        |        |    |   |
| ST_GeometryFr            | ✓ Text         |      |      |        | ✓      |    |   |
| ST_GeomFromI             | ✓              |      |      | ✓      | ✓      |    |   |
| ST_LineFromTe            | ✓              |      |      |        | ✓      |    |   |
| ST_MLineFrom             | ✓ t            |      |      |        | ✓      |    |   |
| ST_MPointFrom            | ✓ ct           |      |      |        | ✓      |    |   |
| ST_MPolyFrom             | ✓ t            |      |      |        | ✓      |    |   |
| ST_PointFromT            | ✓              |      |      |        | ✓      |    |   |
| ST_PolygonFrom           | ✓ ext          |      |      |        | ✓      |    |   |
| ST_WKTToSQL              | ✓              |      |      |        | ✓      |    |   |
| ST_GeogFromWKB           |                | ✓    |      | ✓      |        |    |   |
| ST_GeomFromE             | ✓ CB           |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_GeomFromV             | ✓ 3            |      |      | ✓      | ✓      |    |   |
| ST_LineFromW             | ✓              |      |      |        | ✓      |    |   |
| ST_LinestringF           | ✓ WKB          |      |      |        | ✓      |    |   |
| ST_PointFromW            | ✓              |      | ✓    | ✓      | ✓      |    |   |
| ST_WKBToSQL              | ✓              |      |      |        | ✓      |    |   |
| ST_Box2dFrom             | ✓ Hash         |      |      |        |        |    |   |
| ST_GeomFromC             | ✓ Hash         |      |      |        |        |    |   |
| ST_GeomFromC             | ✓              |      | ✓    |        |        | ✓  | ✓ |
| ST_GeomFromC             | ✓ JSON         |      | ✓    |        |        |    |   |
| ST_GeomFromk             | ✓              |      | ✓    |        |        |    |   |
| ST_GeomFromI             | ✓ CB           |      |      |        |        |    |   |
| ST_GMLToSQL              | ✓              |      |      |        | ✓      |    |   |
| ST_LineFromEn            | ✓ ldedPolyline |      |      |        |        |    |   |
| ST_PointFromGeoHash      |                |      |      |        |        |    |   |
| ST_FromFlatGeobufToTable |                |      |      |        |        |    |   |
| ST_FromFlatGeobuf        |                |      |      |        |        |    |   |
| ST_AsEWKT                | ✓              | ✓    | ✓    | ✓      |        | ✓  | ✓ |
| ST_AsText                | ✓              | ✓    |      | ✓      | ✓      |    |   |
| ST_AsBinary              | ✓              | ✓    | ✓    | ✓      | ✓      | ✓  | ✓ |

| Function             | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|----------------------|------|------|------|--------|--------|----|---|
| ST_AsEWKB            | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_AsHEXEWKB         | ✓    |      | ✓    | ✓      |        |    |   |
| ST_AsEncodedPolyline | ✓    |      |      |        |        |    |   |
| ST_AsFlatGeobuf      | ✗    |      |      |        |        |    |   |
| ST_AsGeobuf          | ✗    |      |      |        |        |    |   |
| ST_AsGeoJSON         | ✓    | ✓    | ✓    |        |        |    |   |
| ST_AsGML             | ✓    | ✓    | ✓    |        | ✓      | ✓  | ✓ |
| ST_AsKML             | ✓    | ✓    | ✓    |        |        |    |   |
| ST_AsLatLonText      | ✓    |      |      |        |        |    |   |
| ST_AsMARC21          | ✓    |      |      |        |        |    |   |
| ST_AsMVTGeom         | ✓    |      |      |        |        |    |   |
| ST_AsMVT             | ✗    |      |      |        |        |    |   |
| ST_AsSVG             | ✓    | ✓    |      | ✓      |        |    |   |
| ST_AsTWKB            | ✓    |      |      |        |        |    |   |
| ST_AsX3D             | ✓    |      | ✓    |        |        | ✓  | ✓ |
| ST_GeoHash           | ✓    |      |      | ✓      |        |    |   |
| &&                   | ✓    | ✓    |      | ✓      |        | ✓  |   |
| &&(geometry,box2df)  | ✓    |      |      | ✓      |        | ✓  |   |
| &&(box2df,geometry)  | ✓    |      |      | ✓      |        | ✓  |   |
| &&(box2df,box2df)    | ✗    |      |      | ✓      |        | ✓  |   |
| &&&                  | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| &&&(geometry,box2df) | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| &&&(gidx,geometry)   | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| &&&(gidx,gidx)       |      |      | ✓    | ✓      |        | ✓  | ✓ |
| &<                   | ✓    |      |      |        |        |    |   |
| &<                   | ✓    |      |      | ✓      |        | ✓  |   |
| &>                   | ✓    |      |      |        |        |    |   |
| <<                   | ✓    |      |      |        |        |    |   |
| <<                   | ✓    |      |      |        |        |    |   |
| =                    | ✓    | ✓    |      | ✓      |        | ✓  |   |
| >>                   | ✓    |      |      |        |        |    |   |
| @                    | ✓    |      |      |        |        |    |   |
| @(geometry,box2df)   | ✓    |      |      | ✓      |        | ✓  |   |
| @(box2df,geometry)   | ✓    |      |      | ✓      |        | ✓  |   |

| Function                 | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|--------------------------|------|------|------|--------|--------|----|---|
| @(box2df,box2df)         | ✓    |      |      | ✓      |        | ✓  |   |
| &>                       | ✓    |      |      |        |        |    |   |
| >>                       | ✓    |      |      |        |        |    |   |
| ~                        | ✓    |      |      |        |        |    |   |
| ~(geometry,box2df)       | ✓    |      |      | ✓      |        | ✓  |   |
| ~(box2df,geometry)       | ✓    |      |      | ✓      |        | ✓  |   |
| ~(box2df,box2df)         | ✓    |      |      | ✓      |        | ✓  |   |
| ~=                       | ✓    |      |      |        |        | ✓  |   |
| <->                      | ✓    | ✓    |      |        |        |    |   |
| =                        | ✓    |      |      |        |        |    |   |
| <#>                      | ✓    |      |      |        |        |    |   |
| <<->>                    | ✓    |      |      |        |        |    |   |
| ST_3DIntersects          | ✓    |      | ✓    |        | ✓      | ✓  | ✓ |
| ST_Contains              | ✓    |      |      |        | ✓      |    |   |
| ST_ContainsProperly      | ✓    |      |      |        |        |    |   |
| ST_CoveredBy             | ✓    | ✓    |      |        |        |    |   |
| ST_Covers                | ✓    | ✓    |      |        |        |    |   |
| ST_Crosses               | ✓    |      |      |        | ✓      |    |   |
| ST_Disjoint              | ✓    |      |      |        | ✓      |    |   |
| ST_Equals                | ✓    |      |      |        | ✓      |    |   |
| ST_Intersects            | ✓    | ✓    |      | ✓      | ✓      |    | ✓ |
| ST_LineCrossingDirection | ✓    |      |      |        |        |    |   |
| ST_OrderingEquals        | ✓    |      |      |        | ✓      |    |   |
| ST_Overlaps              | ✓    |      |      |        | ✓      |    |   |
| ST_Relate                | ✓    |      |      |        | ✓      |    |   |
| ST_RelateMatch           |      |      |      |        |        |    |   |
| ST_Touches               | ✓    |      |      |        | ✓      |    |   |
| ST_Within                | ✓    |      |      |        | ✓      |    |   |
| ST_3DDWithin             | ✓    |      | ✓    |        | ✓      | ✓  |   |
| ST_3DDFullyWithin        | ✓    |      | ✓    |        |        | ✓  |   |
| ST_DFullyWithin          | ✓    |      |      |        |        |    |   |
| ST_DWithin               | ✓    | ✓    |      |        |        |    |   |
| ST_PointInsideCircle     | ✓    |      |      |        |        |    |   |
| ST_Area                  | ✓    | ✓    |      |        | ✓      | ✓  |   |



| Function                | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|-------------------------|------|------|------|--------|--------|----|---|
| ST_Azimuth              | ✓    | ✓    |      |        |        |    |   |
| ST_Angle                | ✓    |      |      |        |        |    |   |
| ST_ClosestPoint         | ✓    | ✓    |      |        |        |    |   |
| ST_3DClosestPoint       | ✓    |      | ✓    |        |        | ✓  |   |
| ST_Distance             | ✓    | ✓    |      | ✓      | ✓      |    |   |
| ST_3DDistance           | ✓    |      | ✓    |        | ✓      | ✓  |   |
| ST_DistanceSphere       | ✓    |      |      |        |        |    |   |
| ST_DistanceSphereoid    | ✓    |      |      |        |        |    |   |
| ST_FrechetDistance      | ✓    |      |      |        |        |    |   |
| ST_HausdorffDistance    | ✓    |      |      |        |        |    |   |
| ST_Length               | ✓    | ✓    |      |        | ✓      |    |   |
| ST_Length2D             | ✓    |      |      |        |        |    |   |
| ST_3DLength             | ✓    |      | ✓    |        | ✓      |    |   |
| ST_LengthSphereoid      | ✓    |      | ✓    |        |        |    |   |
| ST_LongestLine          | ✓    |      |      |        |        |    |   |
| ST_3DLongestLine        | ✓    |      | ✓    |        |        | ✓  |   |
| ST_MaxDistance          | ✓    |      |      |        |        |    |   |
| ST_3DMaxDistance        | ✓    |      | ✓    |        |        | ✓  |   |
| ST_MinimumClearance     | ✓    |      |      |        |        |    |   |
| ST_MinimumClearanceLine | ✓    |      |      |        |        |    |   |
| ST_Perimeter            | ✓    | ✓    |      |        | ✓      |    |   |
| ST_Perimeter2D          | ✓    |      |      |        |        |    |   |
| ST_3DPerimeter          | ✓    |      | ✓    |        | ✓      |    |   |
| ST_ShortestLine         | ✓    | ✓    |      |        |        |    |   |
| ST_3DShortestLine       | ✓    |      | ✓    |        |        | ✓  |   |
| ST_ClipByBox2D          | ✓    |      |      |        |        |    |   |
| ST_Difference           | ✓    |      | ✓    |        | ✓      |    |   |
| ST_Intersection         | ✓    | 😄    | ✓    |        | ✓      |    |   |
| ST_MemUnion             | ✓    |      | ✓    |        |        |    |   |
| ST_Node                 | ✓    |      | ✓    |        |        |    |   |
| ST_Split                | ✓    |      |      |        |        |    |   |
| ST_Subdivide            | ✓    |      |      |        |        |    |   |
| ST_SymDifference        | ✓    |      | ✓    |        | ✓      |    |   |
| ST_UnaryUnion           | ✓    |      | ✓    |        |        |    |   |

| Function                    | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|-----------------------------|------|------|------|--------|--------|----|---|
| ST_Union                    | ✓    |      | ✓    |        | ✓      |    |   |
| ST_Buffer                   | ✓    | 😬    |      |        | ✓      |    |   |
| ST_BuildArea                | ✓    |      |      |        |        |    |   |
| ST_Centroid                 | ✓    | ✓    |      |        | ✓      |    |   |
| ST_ChaikinSmoothing         | ✓    |      | ✓    |        |        |    |   |
| ST_ConcaveHull              | ✓    |      |      |        |        |    |   |
| ST_ConvexHull               | ✓    |      | ✓    |        | ✓      |    |   |
| ST_DelaunayTriangles        | ✓    |      | ✓    |        |        |    | ✓ |
| ST_FilterByM                | ✓    |      |      |        |        |    |   |
| ST_GeneratePoints           | ✓    |      |      |        |        |    |   |
| ST_GeometricMean            | ✓    |      | ✓    |        |        |    |   |
| ST_LineMerge                | ✓    |      |      |        |        |    |   |
| ST_MaximumInscribedCircle   | ✓    |      |      |        |        |    |   |
| ST_LargestEmptyCircle       | ✓    |      |      |        |        |    |   |
| ST_MinimumBoundingCircle    | ✓    |      |      |        |        |    |   |
| ST_MinimumBoundingRadius    | ✓    |      |      |        |        |    |   |
| ST_OrientedEnvelope         | ✓    |      |      |        |        |    |   |
| ST_OffsetCurve              | ✓    |      |      |        |        |    |   |
| ST_PointOnSurface           | ✓    |      | ✓    |        | ✓      |    |   |
| ST_Polygonize               | ✓    |      |      |        |        |    |   |
| ST_ReducePrecision          | ✓    |      |      |        |        |    |   |
| ST_SharedPaths              | ✓    |      |      |        |        |    |   |
| ST_Simplify                 | ✓    |      |      |        |        |    |   |
| ST_SimplifyPreserveTopology | ✓    |      |      |        |        |    |   |
| ST_SimplifyPolygonHull      | ✓    |      |      |        |        |    |   |
| ST_SimplifyVW               | ✓    |      |      |        |        |    |   |
| ST_SetEffectiveArea         | ✓    |      |      |        |        |    |   |
| ST_TriangulatePolygon       | ✓    |      |      |        |        |    |   |
| ST_VoronoiLines             | ✓    |      |      |        |        |    |   |
| ST_VoronoiPolygons          | ✓    |      |      |        |        |    |   |
| ST_CoverageInvalidEdges     | ✓    |      |      |        |        |    |   |
| ST_CoverageSimplify         | ✓    |      |      |        |        |    |   |
| ST_CoverageUnion            | ✓    |      |      |        |        |    |   |
| ST_CoverageClean            | ✓    |      |      |        |        |    |   |

| Function                  | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|---------------------------|------|------|------|--------|--------|----|---|
| ST_Affine                 | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_Rotate                 | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_RotateX                | ✓    |      | ✓    |        |        | ✓  | ✓ |
| ST_RotateY                | ✓    |      | ✓    |        |        | ✓  | ✓ |
| ST_RotateZ                | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_Scale                  | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_Translate              | ✓    |      | ✓    | ✓      |        |    |   |
| ST_TransScale             | ✓    |      | ✓    | ✓      |        |    |   |
| ST_ClusterDBSCAN          | ✓    |      |      | ✓      |        |    |   |
| ST_ClusterIntersecting    | ✓    |      |      |        |        |    |   |
| ST_ClusterIntersectingWin | ✓    |      |      |        |        |    |   |
| ST_ClusterKMeans          | ✓    |      |      |        |        |    |   |
| ST_ClusterWithin          | ✓    |      |      | ✓      |        |    |   |
| ST_ClusterWithin/in       | ✓    |      |      | ✓      |        |    |   |
| Box2D                     | ✓    |      |      | ✓      |        | ✓  | ✓ |
| Box3D                     | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_EstimatedExtent        | ✗    |      |      | ✓      |        |    |   |
| ST_Expand                 | ✓    |      |      |        |        | ✓  | ✓ |
| ST_Extent                 | ✓    |      |      |        |        | ✓  | ✓ |
| ST_3DExtent               | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_MakeBox2D              | ✓    |      |      |        |        |    |   |
| ST_3DMakeBox              | ✓    |      |      |        |        |    |   |
| ST_XMax                   | ✗    |      | ✓    | ✓      |        |    |   |
| ST_XMin                   | ✗    |      | ✓    | ✓      |        |    |   |
| ST_YMax                   | ✗    |      | ✓    | ✓      |        |    |   |
| ST_YMin                   | ✗    |      | ✓    | ✓      |        |    |   |
| ST_ZMax                   | ✗    |      | ✓    | ✓      |        |    |   |
| ST_ZMin                   | ✗    |      | ✓    | ✓      |        |    |   |
| ST_LineInterpolatePoint   | ✓    | ✓    | ✓    |        |        |    |   |
| ST_3DLineInterpolatePoint | ✓    |      | ✓    |        |        |    |   |
| ST_LineInterpolatePoints  | ✓    | ✓    | ✓    |        |        |    |   |
| ST_LineLocatePoint        | ✓    | ✓    |      |        |        |    |   |
| ST_LineSubstring          | ✓    | ✓    | ✓    |        |        |    |   |
| ST_LocateAlong            | ✓    |      |      |        | ✓      |    |   |

| Function                     | geom                                                                                | geog | 2.5D                                                                                | Curves | SQL MM                                                                                | PS                                                                                    | T                                                                                     |
|------------------------------|-------------------------------------------------------------------------------------|------|-------------------------------------------------------------------------------------|--------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| ST_LocateBetween             | ✓                                                                                   |      |                                                                                     |        | ✓                                                                                     |                                                                                       |                                                                                       |
| ST_LocateBetweenElevations   | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                       |                                                                                       |                                                                                       |
| ST_InterpolateFrom           | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                       |                                                                                       |                                                                                       |
| ST_AddMeasure                | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                       |                                                                                       |                                                                                       |
| ST_IsValidTrajectory         | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                       |                                                                                       |                                                                                       |
| ST_ClosestPointApproach      | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                       |                                                                                       |                                                                                       |
| ST_DistanceCPA               | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                       |                                                                                       |                                                                                       |
| ST_CPAWithin                 | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                       |                                                                                       |                                                                                       |
| postgis.gdal_datapath        |                                                                                     |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| postgis.gdal_enabled_drivers |                                                                                     |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| postgis.enable_outdb_rasters |                                                                                     |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| postgis.gdal_vsi_options     |                                                                                     |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| postgis.gdal_cpl_debug       |                                                                                     |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| PostGIS_AddBB                | ✓                                                                                   |      |                                                                                     | ✓      |                                                                                       |                                                                                       |                                                                                       |
| PostGIS_DropBB               | ✓                                                                                   |      |                                                                                     | ✓      |                                                                                       |                                                                                       |                                                                                       |
| PostGIS_HasBB                | ✓                                                                                   |      |                                                                                     | ✓      |                                                                                       |                                                                                       |                                                                                       |
| postgis_sfcgal_version       |                                                                                     |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| postgis_sfcgal_full_version  |                                                                                     |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| CG_ForceLHR                  |  |      |  |        |                                                                                       |  |  |
| CG_IsPlanar                  |  |      |  |        |                                                                                       |  |  |
| CG_IsSolid                   |  |      |  |        |                                                                                       |  |  |
| CG_MakeSolid                 |  |      |  |        |                                                                                       |  |  |
| CG_Orientation               |  |      |  |        |                                                                                       |                                                                                       |                                                                                       |
| CG_Area                      |  |      |                                                                                     |        |                                                                                       |                                                                                       |                                                                                       |
| CG_3DArea                    |  |      |  |        |  |  |  |
| CG_Volume                    |  |      |  |        |  |  |  |
| ST_ForceLHR                  |  |      |  |        |                                                                                       |  |  |
| ST_IsPlanar                  |  |      |  |        |                                                                                       |  |  |
| ST_IsSolid                   |  |      |  |        |                                                                                       |  |  |
| ST_MakeSolid                 |  |      |  |        |                                                                                       |  |  |
| ST_Orientation               |  |      |  |        |                                                                                       |                                                                                       |                                                                                       |

| Function        | geom | geog            | 2.5D | Curves | SQL MM | PS | T |
|-----------------|------|-----------------|------|--------|--------|----|---|
| ST_3DArea       |      |                 |      |        |        |    |   |
| ST_Volume       |      |                 |      |        |        |    |   |
| CG_Intersection |      |                 |      |        |        |    |   |
| CG_Intersects   |      |                 |      |        |        |    |   |
| CG_3DIntersec   |      |                 |      |        |        |    |   |
| CG_Difference   |      |                 |      |        |        |    |   |
| ST_3DDifferenc  |      |                 |      |        |        |    |   |
| CG_3DDifferenc  |      |                 |      |        |        |    |   |
| CG_Distance     |      |                 |      |        |        |    |   |
| CG_3DDistance   |      |                 |      |        |        |    |   |
| ST_3DConvexH    |      |                 |      |        |        |    |   |
| CG_3DConvexH    |      |                 |      |        |        |    |   |
| ST_3DIntersect  |      |                 |      |        |        |    |   |
| CG_3DIntersec   |      |                 |      |        |        |    |   |
| CG_Union        |      |                 |      |        |        |    |   |
| ST_3DUnion      |      |                 |      |        |        |    |   |
| CG_3DUnion      |      |                 |      |        |        |    |   |
| ST_AlphaShape✓  |      |                 |      |        |        |    |   |
| CG_AlphaShape   |      |                 |      |        |        |    |   |
| CG_ApproxCon    |      | Partition       |      |        |        |    |   |
| ST_Approximat   |      | edialAxis       |      |        |        |    |   |
| CG_Approximat   |      | edialAxis       |      |        |        |    |   |
| ST_Constrained  |      | launayTriangles |      |        |        |    |   |
| CG_Constrained  |      | launayTriangles |      |        |        |    |   |

| Function                       | geom                                                                                | geog | 2.5D                                                                                | Curves | SQL MM | PS                                                                                    | T                                                                                     |
|--------------------------------|-------------------------------------------------------------------------------------|------|-------------------------------------------------------------------------------------|--------|--------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| ST_Extrude                     |    |      |    |        |        |    |    |
| CG_Extrude                     |    |      |    |        |        |    |    |
| CG_ExtrudeStraightSkeleton     |    |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| CG_GreeneApproxConvexPartition |    |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_MinkowskiSum                |    |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| CG_MinkowskiSum                |    |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_OptimalAlphaShape           |    |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| CG_OptimalAlphaShape           |    |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| CG_OptimalCorrelationPartition |    |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| CG_StraightSkeleton            |   |      |   |        |        |   |   |
| ST_StraightSkeleton            |  |      |  |        |        |  |  |
| ST_Tessellate                  |  |      |  |        |        |  |  |
| CG_Tessellate                  |  |      |  |        |        |  |  |
| CG_Triangulate                 |  |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| CG_Visibility                  |  |      |  |        |        |  |  |
| CG_YMonotonePartition          |  |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| CG_StraightSkeletonPartition   |  |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| CG_3DBuffer                    |  |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| CG_Rotate                      |  |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| CG_2DRotate                    |  |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| CG_3DRotate                    |  |      |  |        |        |                                                                                       |                                                                                       |
| CG_RotateX                     |  |      |  |        |        |                                                                                       |                                                                                       |
| CG_RotateY                     |  |      |  |        |        |                                                                                       |                                                                                       |
| CG_RotateZ                     |  |      |  |        |        |                                                                                       |                                                                                       |

| Function                     | geom                                                                              | geog | 2.5D                                                                              | Curves | SQL MM | PS | T |
|------------------------------|-----------------------------------------------------------------------------------|------|-----------------------------------------------------------------------------------|--------|--------|----|---|
| CG_Scale                     |  |      |                                                                                   |        |        |    |   |
| CG_3DScale                   |  |      |  |        |        |    |   |
| CG_3DScaleArcCenter          |  |      |  |        |        |    |   |
| CG_Translate                 |  |      |                                                                                   |        |        |    |   |
| CG_3DTranslate               |  |      |  |        |        |    |   |
| CG_Simplify                  |  |      |  |        |        |    |   |
| CG_3DAlphaWrapping           |  |      |  |        |        |    |   |
| getfaceedges_returntype      |                                                                                   |      |                                                                                   |        |        |    |   |
| TopoGeometry                 |                                                                                   |      |                                                                                   |        |        |    |   |
| validate_topology_returntype |                                                                                   |      |                                                                                   |        |        |    |   |
| TopoElement                  |                                                                                   |      |                                                                                   |        |        |    |   |
| TopoElementArray             |                                                                                   |      |                                                                                   |        |        |    |   |
| AddTopoGeometryColumn        |                                                                                   |      |                                                                                   |        |        |    |   |
| RenameTopoGeometryColumn     |                                                                                   |      |                                                                                   |        |        |    |   |
| DropTopology                 |                                                                                   |      |                                                                                   |        |        |    |   |
| RenameTopology               |                                                                                   |      |                                                                                   |        |        |    |   |
| DropTopoGeometryColumn       |                                                                                   |      |                                                                                   |        |        |    |   |
| Populate_Topology_Layer      |                                                                                   |      |                                                                                   |        |        |    |   |
| TopologySummary              |                                                                                   |      |                                                                                   |        |        |    |   |
| ValidateTopology             | ✓                                                                                 |      |                                                                                   |        |        |    |   |
| ValidateTopologyRelation     |                                                                                   |      |                                                                                   |        |        |    |   |
| ValidateTopologyPrecision    | ✓                                                                                 |      |                                                                                   |        |        |    |   |
| MakeTopologyPersistent       | ✓                                                                                 |      |                                                                                   |        |        |    |   |
| FindTopology                 | ✓                                                                                 |      |                                                                                   |        |        |    |   |
| FindLayer                    | ✓                                                                                 |      |                                                                                   |        |        |    |   |
| TotalTopologySize            |                                                                                   |      |                                                                                   |        |        |    |   |
| UpgradeTopology              |                                                                                   |      |                                                                                   |        |        |    |   |
| CreateTopology               |                                                                                   |      |                                                                                   |        |        |    |   |
| CopyTopology                 |                                                                                   |      |                                                                                   |        |        |    |   |
| ST_InitTopoGeo               |                                                                                   |      |                                                                                   |        | ✓      |    |   |
| ST_CreateTopoGeo             | ✓                                                                                 |      |                                                                                   |        | ✓      |    |   |
| TopoGeo_AddPoint             | ✓                                                                                 |      |                                                                                   |        |        |    |   |
| TopoGeo_AddLineString        | ✓                                                                                 |      |                                                                                   |        |        |    |   |
| TopoGeo_AddPolygon           | ✓                                                                                 |      |                                                                                   |        |        |    |   |
| TopoGeo_LoadGeometry         | ✓                                                                                 |      |                                                                                   |        |        |    |   |
| ST_AddIsoNode                | ✓                                                                                 |      |                                                                                   |        | ✓      |    |   |
| ST_AddIsoEdge                | ✓                                                                                 |      |                                                                                   |        | ✓      |    |   |
| ST_AddEdgeNeighbours         | ✓                                                                                 |      |                                                                                   |        | ✓      |    |   |

| Function             | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|----------------------|------|------|------|--------|--------|----|---|
| ST_AddEdgeModFace    | ✓    |      |      |        | ✓      |    |   |
| ST_RemEdgeNewFace    |      |      |      |        | ✓      |    |   |
| ST_RemEdgeModFace    |      |      |      |        | ✓      |    |   |
| ST_ChangeEdgeGeom    | ✓    |      |      |        | ✓      |    |   |
| ST_ModEdgeSplit      | ✓    |      |      |        | ✓      |    |   |
| ST_ModEdgeHeal       |      |      |      |        | ✓      |    |   |
| ST_NewEdgeHeal       |      |      |      |        | ✓      |    |   |
| ST_MoveIsoNode       | ✓    |      |      |        | ✓      |    |   |
| ST_NewEdgesSplit     | ✓    |      |      |        | ✓      |    |   |
| ST_RemoveIsoNode     |      |      |      |        | ✓      |    |   |
| ST_RemoveIsoEdge     |      |      |      |        | ✓      |    |   |
| GetEdgeByPoint       | ✓    |      |      |        |        |    |   |
| GetFaceByPoint       | ✓    |      |      |        |        |    |   |
| GetFaceContainsPoint | ✓    |      |      |        |        |    |   |
| GetNodeByPoint       | ✓    |      |      |        |        |    |   |
| GetTopologyID        |      |      |      |        |        |    |   |
| GetTopologySRID      |      |      |      |        |        |    |   |
| GetTopologyName      |      |      |      |        |        |    |   |
| ST_GetFaceEdges      |      |      |      |        | ✓      |    |   |
| ST_GetFaceGeometry   | ✓    |      |      |        | ✓      |    |   |
| GetRingEdges         |      |      |      |        |        |    |   |
| GetNodeEdges         |      |      |      |        |        |    |   |
| Polygonize           |      |      |      |        |        |    |   |
| AddNode              | ✓    |      |      |        |        |    |   |
| AddEdge              | ✓    |      |      |        |        |    |   |
| AddFace              | ✓    |      |      |        |        |    |   |
| ST_Simplify          | ✓    |      |      |        |        |    |   |
| RemoveUnusedNodes    | ✓    |      |      |        |        |    |   |
| CreateTopoGeom       | ✓    |      |      |        |        |    |   |
| toTopoGeom           | ✓    |      |      |        |        |    |   |
| TopoElementArray_Agg |      |      |      |        |        |    |   |
| TopoElement          | ✓    |      |      |        |        |    |   |
| clearTopoGeom        | ✓    |      |      |        |        |    |   |
| TopoGeom_addNode     | ✓    |      |      |        |        |    |   |
| TopoGeom_removeNode  | ✓    |      |      |        |        |    |   |
| TopoGeom_addFace     | ✓    |      |      |        |        |    |   |
| toTopoGeom           |      |      |      |        |        |    |   |



| Function                      | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|-------------------------------|------|------|------|--------|--------|----|---|
| GetTopoGeomElementArray       |      |      |      |        |        |    |   |
| GetTopoGeomElements           |      |      |      |        |        |    |   |
| ST_SRID                       | ✓    |      |      |        | ✓      |    |   |
| AsGML                         | ✓    |      |      |        |        |    |   |
| AsTopoJSON                    | ✓    |      |      |        |        |    |   |
| Equals                        | ✓    |      | ✓    |        |        |    |   |
| Intersects                    | ✓    |      | ✓    |        |        |    |   |
| geomval                       |      |      |      |        |        |    |   |
| addbandarg                    |      |      |      |        |        |    |   |
| rastbandarg                   |      |      |      |        |        |    |   |
| raster                        |      |      |      |        |        |    |   |
| reclassarg                    |      |      |      |        |        |    |   |
| summarystats                  |      |      |      |        |        |    |   |
| unionarg                      |      |      |      |        |        |    |   |
| AddRasterConstraints          |      |      |      |        |        |    |   |
| DropRasterConstraints         |      |      |      |        |        |    |   |
| AddOverviewConstraints        |      |      |      |        |        |    |   |
| DropOverviewConstraints       |      |      |      |        |        |    |   |
| PostGIS_GDAL_Version          |      |      |      |        |        |    |   |
| PostGIS_Raster_Lib_Build_Date |      |      |      |        |        |    |   |
| PostGIS_Raster_Lib_Version    |      |      |      |        |        |    |   |
| ST_GDALDrivers                |      |      |      |        |        |    |   |
| UpdateRasterSRID              |      |      |      |        |        |    |   |
| ST_CreateOverview             |      |      |      |        |        |    |   |
| ST_AddBand                    |      |      |      |        |        |    |   |
| ST_AsRaster                   | ✓    |      |      |        |        |    |   |
| ST_AsRasterAgg                | ✓    |      |      |        |        |    |   |
| ST_Band                       |      |      |      |        |        |    |   |
| ST_MakeEmptyCoverage          |      |      |      |        |        |    |   |
| ST_MakeEmptyRaster            |      |      |      |        |        |    |   |
| ST_Tile                       |      |      |      |        |        |    |   |
| ST_Retile                     | ✓    |      |      |        |        |    |   |
| ST_FromGDALRaster             |      |      |      |        |        |    |   |
| ST_GeoReference               |      |      |      |        |        |    |   |
| ST_Height                     |      |      |      |        |        |    |   |
| ST_IsEmpty                    |      |      |      |        |        |    |   |
| ST_MemSize                    |      |      |      |        |        |    |   |
| ST_MetaData                   |      |      |      |        |        |    |   |
| ST_NumBands                   |      |      |      |        |        |    |   |
| ST_PixelHeight                |      |      |      |        |        |    |   |
| ST_PixelWidth                 |      |      |      |        |        |    |   |
| ST_ScaleX                     |      |      |      |        |        |    |   |
| ST_ScaleY                     |      |      |      |        |        |    |   |
| ST_RasterToWorldCoord         |      |      |      |        |        |    |   |
| ST_RasterToWorldCoordX        |      |      |      |        |        |    |   |
| ST_RasterToWorldCoordY        |      |      |      |        |        |    |   |
| ST_Rotation                   |      |      |      |        |        |    |   |
| ST_SkewX                      |      |      |      |        |        |    |   |
| ST_SkewY                      |      |      |      |        |        |    |   |
| ST_SRID                       |      |      |      |        |        |    |   |
| ST_Summary                    |      |      |      |        |        |    |   |

| Function              | geom     | geog | 2.5D | Curves | SQL MM | PS | T |
|-----------------------|----------|------|------|--------|--------|----|---|
| ST_UpperLeftX         |          |      |      |        |        |    |   |
| ST_UpperLeftY         |          |      |      |        |        |    |   |
| ST_Width              |          |      |      |        |        |    |   |
| ST_WorldToRas         | ✓ Coord  |      |      |        |        |    |   |
| ST_WorldToRas         | ✓ CoordX |      |      |        |        |    |   |
| ST_WorldToRas         | ✓ CoordY |      |      |        |        |    |   |
| ST_BandMetaData       |          |      |      |        |        |    |   |
| ST_BandNoDataValue    |          |      |      |        |        |    |   |
| ST_BandIsNoData       |          |      |      |        |        |    |   |
| ST_BandPath           |          |      |      |        |        |    |   |
| ST_BandFileSize       |          |      |      |        |        |    |   |
| ST_BandFileTimestamp  |          |      |      |        |        |    |   |
| ST_BandPixelType      |          |      |      |        |        |    |   |
| ST_MinPossibleValue   |          |      |      |        |        |    |   |
| ST_HasNoBand          |          |      |      |        |        |    |   |
| ST_PixelAsPoly        | ✓        |      |      |        |        |    |   |
| ST_PixelAsPolygons    |          |      |      |        |        |    |   |
| ST_PixelAsPoint       | ✓        |      |      |        |        |    |   |
| ST_PixelAsPoints      |          |      |      |        |        |    |   |
| ST_PixelAsCent        | ✓ 1      |      |      |        |        |    |   |
| ST_PixelAsCentroids   |          |      |      |        |        |    |   |
| ST_Value              | ✓        |      |      |        |        |    |   |
| ST_NearestValue       | ✓        |      |      |        |        |    |   |
| ST_SetZ               | ✓        |      |      |        |        |    |   |
| ST_SetM               | ✓        |      |      |        |        |    |   |
| ST_Neighborhood       | ✓        |      |      |        |        |    |   |
| ST_SetValue           | ✓        |      |      |        |        |    |   |
| ST_SetValues          |          |      |      |        |        |    |   |
| ST_DumpValues         |          |      |      |        |        |    |   |
| ST_PixelOfValue       |          |      |      |        |        |    |   |
| ST_SetGeoReference    |          |      |      |        |        |    |   |
| ST_SetRotation        |          |      |      |        |        |    |   |
| ST_SetScale           |          |      |      |        |        |    |   |
| ST_SetSkew            |          |      |      |        |        |    |   |
| ST_SetSRID            |          |      |      |        |        |    |   |
| ST_SetUpperLeft       |          |      |      |        |        |    |   |
| ST_Resample           |          |      |      |        |        |    |   |
| ST_Rescale            |          |      |      |        |        |    |   |
| ST_Reskew             |          |      |      |        |        |    |   |
| ST_SnapToGrid         |          |      |      |        |        |    |   |
| ST_Resize             |          |      |      |        |        |    |   |
| ST_Transform          |          |      |      |        |        |    |   |
| ST_SetBandNoDataValue |          |      |      |        |        |    |   |
| ST_SetBandIsNoData    |          |      |      |        |        |    |   |
| ST_SetBandPath        |          |      |      |        |        |    |   |
| ST_SetBandIndex       |          |      |      |        |        |    |   |
| ST_Count              |          |      |      |        |        |    |   |
| ST_CountAgg           |          |      |      |        |        |    |   |

| Function                                           | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|----------------------------------------------------|------|------|------|--------|--------|----|---|
| ST_Histogram                                       |      |      |      |        |        |    |   |
| ST_Quantile                                        |      |      |      |        |        |    |   |
| ST_SummaryStats                                    |      |      |      |        |        |    |   |
| ST_SummaryStatsAgg                                 |      |      |      |        |        |    |   |
| ST_ValueCount                                      |      |      |      |        |        |    |   |
| ST_RastFromWKB                                     |      |      |      |        |        |    |   |
| ST_RastFromHexWKB                                  |      |      |      |        |        |    |   |
| ST_AsBinary/ST_AsWKB                               |      |      |      |        |        |    |   |
| ST_AsHexWKB                                        |      |      |      |        |        |    |   |
| ST_AsGDALRaster                                    |      |      |      |        |        |    |   |
| ST_AsJPEG                                          |      |      |      |        |        |    |   |
| ST_AsPNG                                           |      |      |      |        |        |    |   |
| ST_AsTIFF                                          |      |      |      |        |        |    |   |
| ST_Clip                                            | ✓    |      |      |        |        |    |   |
| ST_ColorMap                                        |      |      |      |        |        |    |   |
| ST_Grayscale                                       |      |      |      |        |        |    |   |
| ST_Intersection                                    | ✓    |      |      |        |        |    |   |
| ST_MapAlgebra<br>(callback<br>function<br>version) |      |      |      |        |        |    |   |
| ST_MapAlgebra<br>(expres-<br>sion<br>version)      |      |      |      |        |        |    |   |
| ST_MapAlgebraExpr                                  |      |      |      |        |        |    |   |
| ST_MapAlgebraExpr                                  |      |      |      |        |        |    |   |
| ST_MapAlgebraFct                                   |      |      |      |        |        |    |   |
| ST_MapAlgebraFct                                   |      |      |      |        |        |    |   |
| ST_MapAlgebraFctNgb                                |      |      |      |        |        |    |   |
| ST_Reclass                                         |      |      |      |        |        |    |   |
| ST_ReclassExact                                    |      |      |      |        |        |    |   |
| ST_Union                                           |      |      |      |        |        |    |   |
| ST_Distinct4ma                                     |      |      |      |        |        |    |   |
| ST_InvDistWeight4ma                                |      |      |      |        |        |    |   |
| ST_Max4ma                                          |      |      |      |        |        |    |   |
| ST_Mean4ma                                         |      |      |      |        |        |    |   |
| ST_Min4ma                                          |      |      |      |        |        |    |   |
| ST_MinDist4ma                                      |      |      |      |        |        |    |   |
| ST_Range4ma                                        |      |      |      |        |        |    |   |
| ST_StdDev4ma                                       |      |      |      |        |        |    |   |
| ST_Sum4ma                                          |      |      |      |        |        |    |   |
| ST_Aspect                                          |      |      |      |        |        |    |   |
| ST_HillShade                                       |      |      |      |        |        |    |   |
| ST_Roughness                                       |      |      |      |        |        |    |   |
| ST_Slope                                           |      |      |      |        |        |    |   |
| ST_TPI                                             |      |      |      |        |        |    |   |
| ST_TRI                                             |      |      |      |        |        |    |   |
| ST_InterpolateF                                    | ✓    |      |      |        |        |    |   |
| ST_Contour                                         |      |      |      |        |        |    |   |
| Box3D                                              | ✓    |      |      |        |        |    |   |
| ST_ConvexHull                                      | ✓    |      |      |        |        |    |   |
| ST_DumpAsPolygons                                  |      |      |      |        |        |    |   |

| Function                           | geom | geog   | 2.5D | Curves | SQL MM | PS | T |
|------------------------------------|------|--------|------|--------|--------|----|---|
| ST_Envelope                        | ✓    |        |      |        |        |    |   |
| ST_MinConvexHull                   | ✓    |        |      |        |        |    |   |
| ST_Polygon                         | ✓    |        |      |        |        |    |   |
| ST_Intersection                    | ✓    | ctions |      |        |        |    |   |
| &&                                 | ✓    |        |      |        |        |    |   |
| &<                                 |      |        |      |        |        |    |   |
| &>                                 |      |        |      |        |        |    |   |
| =                                  |      |        |      |        |        |    |   |
| @                                  | ✓    |        |      |        |        |    |   |
| ~ =                                |      |        |      |        |        |    |   |
| ~                                  | ✓    |        |      |        |        |    |   |
| ST_Contains                        |      |        |      |        |        |    |   |
| ST_ContainsProperly                |      |        |      |        |        |    |   |
| ST_Covers                          |      |        |      |        |        |    |   |
| ST_CoveredBy                       |      |        |      |        |        |    |   |
| ST_Disjoint                        |      |        |      |        |        |    |   |
| ST_Intersects                      | ✓    |        |      |        |        |    |   |
| ST_Overlaps                        |      |        |      |        |        |    |   |
| ST_Touches                         |      |        |      |        |        |    |   |
| ST_SameAlignment                   |      |        |      |        |        |    |   |
| ST_NotSameAlignmentReason          |      |        |      |        |        |    |   |
| ST_Within                          |      |        |      |        |        |    |   |
| ST_DWithin                         |      |        |      |        |        |    |   |
| ST_DFullyWithin                    |      |        |      |        |        |    |   |
| stdaddr                            |      |        |      |        |        |    |   |
| rules                              |      |        |      |        |        |    |   |
| table                              |      |        |      |        |        |    |   |
| lex table                          |      |        |      |        |        |    |   |
| gaz table                          |      |        |      |        |        |    |   |
| debug_standardize_address          |      |        |      |        |        |    |   |
| parse_address                      |      |        |      |        |        |    |   |
| standardize_address                |      |        |      |        |        |    |   |
| Drop_Indexes_Generate_Script       |      |        |      |        |        |    |   |
| Drop_Nation_Tables_Generate_Script |      |        |      |        |        |    |   |
| Drop_State_Tables_Generate_Script  |      |        |      |        |        |    |   |
| Geocode                            | ✓    |        |      |        |        |    |   |
| Geocode_Intersections              | ✓    |        |      |        |        |    |   |
| Get_Geocode_Setting                |      |        |      |        |        |    |   |
| Get_Tract                          | ✓    |        |      |        |        |    |   |
| Install_Missing_Indexes            |      |        |      |        |        |    |   |
| Loader_Generate_Census_Script      |      |        |      |        |        |    |   |
| Loader_Generate_Script             |      |        |      |        |        |    |   |
| Loader_Generate_Nation_Script      |      |        |      |        |        |    |   |
| Missing_Indexes_Generate_Script    |      |        |      |        |        |    |   |
| Normalize_Address                  |      |        |      |        |        |    |   |
| Pgc_Normalize_Address              |      |        |      |        |        |    |   |
| Pprint_Addy                        |      |        |      |        |        |    |   |
| Reverse_Geocode                    | ✓    |        |      |        |        |    |   |

| Function                            | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|-------------------------------------|------|------|------|--------|--------|----|---|
| <a href="#">Topology_Load_Tiger</a> |      |      |      |        |        |    |   |
| <a href="#">Set_Geocode_Setting</a> |      |      |      |        |        |    |   |

## 13.12 New, Enhanced or changed PostGIS Functions

### 13.12.1 PostGIS Functions new or enhanced in 3.6

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.6

- **CG\_2DRotate** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry by a given angle around a specified point in 2D.
- **CG\_3DAlphaWrapping** - Availability: 3.6.0 - requires SFCGAL >= 2.1.0 Computes a 3D Alpha-wrapping strictly enclosing a geometry.
- **CG\_3DBuffer** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Computes a 3D buffer around a geometry.
- **CG\_3DRotate** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry in 3D space around an axis vector.
- **CG\_3DScale** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Scales a geometry by separate factors along X, Y, and Z axes.
- **CG\_3DScaleAroundCenter** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Scales a geometry in 3D space around a specified center point.
- **CG\_3DTranslate** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Translates (moves) a geometry by given offsets in 3D space.
- **CG\_Rotate** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry by a given angle around the origin (0,0).
- **CG\_RotateX** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry around the X-axis by a given angle.
- **CG\_RotateY** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry around the Y-axis by a given angle.
- **CG\_RotateZ** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry around the Z-axis by a given angle.
- **CG\_Scale** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Scales a geometry uniformly in all dimensions by a given factor.
- **CG\_Simplify** - Availability: 3.6.0 - requires SFCGAL >= 2.1.0 Reduces the complexity of a geometry while preserving essential features and Z/M values.
- **CG\_StraightSkeletonPartition** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0. Computes the straight skeleton partition of a polygon.
- **CG\_Translate** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Translates (moves) a geometry by given offsets in 2D space.
- **MakeTopologyPrecise** - Availability: 3.6.0 Snap topology vertices to precision grid.
- **ST\_AsRasterAgg** - Availability: 3.6.0 Aggregate. Renders PostGIS geometries into a new raster.

- **ST\_CoverageClean** - Availability: 3.6.0 - requires GEOS >= 3.14.0 Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.
- **ST\_IntersectionFractions** - Availability: 3.6.0 Requires GEOS 3.14 or higher. Calculates the fraction of each raster cell that is covered by a given geometry.
- **ST\_ReclassExact** - Availability: 3.6.0 Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.
- **TotalTopologySize** - Availability: 3.6.0 Total disk space used by the specified topology, including all indexes and TOAST data.
- **UpgradeTopology** - Availability: 3.6.0 Upgrades the specified topology to support large ids (int8) for topology and primitive ids.
- **ValidateTopologyPrecision** - Availability: 3.6.0 Returns non-precise vertices in the topology.
- **postgis.gdal\_cpl\_debug** - Availability: 3.6.0 A boolean configuration to turn logging of GDAL debug messages on and off.

### 13.12.2 PostGIS Functions new or enhanced in 3.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.5

- **CG\_3DArea** - Availability: 3.5.0 3 **geometry** **geometry** **geometry**. **geometry** 0 **geometry**.
- **CG\_3DConvexHull** - Availability: 3.5.0 **geometry** **geometry**.
- **CG\_3DDifference** - Availability: 3.5.0 3 **geometry** **geometry**.
- **CG\_3DDistance** - Availability: 3.5.0 Computes the minimum 3D distance between two geometries
- **CG\_3DIntersection** - Availability: 3.5.0 3 **geometry** **geometry** **geometry**.
- **CG\_3DIntersects** - Availability: 3.5.0 Tests if two 3D geometries intersect
- **CG\_3DUnion** - Availability: 3.5.0 Perform 3D union using postgis\_sfcgal.
- **CG\_AlphaShape** - Availability: 3.5.0 - requires SFCGAL >= 1.4.1. Computes an Alpha-shape enclosing a geometry
- **CG\_ApproxConvexPartition** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Computes approximal convex partition of the polygon geometry
- **CG\_ApproximateMedialAxis** - Availability: 3.5.0 **geometry** **geometry** **geometry**.
- **CG\_Area** - Availability: 3.5.0 Calculates the area of a geometry
- **CG\_Difference** - Availability: 3.5.0 Computes the geometric difference between two geometries
- **CG\_Distance** - Availability: 3.5.0 Computes the minimum distance between two geometries
- **CG\_Extrude** - Availability: 3.5.0 **geometry** **geometry** **geometry**.
- **CG\_ExtrudeStraightSkeleton** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Straight Skeleton Extrusion
- **CG\_ForceLHR** - Availability: 3.5.0 LHR(Left Hand Reverse; **geometry**) **geometry**.
- **CG\_GreeneApproxConvexPartition** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Computes approximal convex partition of the polygon geometry

- **CG\_Intersection** - Availability: 3.5.0 Computes the intersection of two geometries
- **CG\_Intersects** - Availability: 3.5.0 Tests if two geometries intersect (they have at least one point in common)
- **CG\_IsPlanar** - Availability: 3.5.0 `boolean isPlanar(Geometry geometry)`.
- **CG\_IsSolid** - Availability: 3.5.0 `boolean isSolid(Geometry geometry)`. `boolean isSolid(Geometry geometry, boolean isShell)`.
- **CG\_MakeSolid** - Availability: 3.5.0 `Geometry makeSolid(Geometry geometry)`. `Geometry makeSolid(Geometry geometry, boolean isShell)`. `Geometry makeSolid(Geometry geometry, boolean isShell, boolean isTIN)` TIN `boolean isTIN(Geometry geometry)`.
- **CG\_MinkowskiSum** - Availability: 3.5.0 `Geometry minkowskiSum(Geometry geometry, Geometry geometry2)`.
- **CG\_OptimalAlphaShape** - Availability: 3.5.0 - requires SFCGAL >= 1.4.1. Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.
- **CG\_OptimalConvexPartition** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Computes an optimal convex partition of the polygon geometry
- **CG\_Orientation** - Availability: 3.5.0 `int orientation(Geometry geometry)` (orientation) `int orientation(Geometry geometry, int orientation)`.
- **CG\_StraightSkeleton** - Availability: 3.5.0 `Geometry straightSkeleton(Geometry geometry)` (straight skeleton) `Geometry straightSkeleton(Geometry geometry, int orientation)`.
- **CG\_Tessellate** - Availability: 3.5.0 `Geometry tessellate(Geometry geometry)` (tessellation) `Geometry tessellate(Geometry geometry, int orientation)` TIN `boolean isTIN(Geometry geometry)` TIN `boolean isTIN(Geometry geometry)`.
- **CG\_Triangulate** - Availability: 3.5.0 Triangulates a polygonal geometry
- **CG\_Union** - Availability: 3.5.0 Computes the union of two geometries
- **CG\_Visibility** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Compute a visibility polygon from a point or a segment in a polygon geometry
- **CG\_Volume** - Availability: 3.5.0 3 `double volume(Geometry geometry)`. `double volume(Geometry geometry, int orientation)` (orientation) 0 `double volume(Geometry geometry)`.
- **CG\_YMonotonePartition** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Computes y-monotone partition of the polygon geometry
- **ST\_HasM** - Availability: 3.5.0 Checks if a geometry has an M (measure) dimension.
- **ST\_HasZ** - Availability: 3.5.0 Checks if a geometry has a Z dimension.
- **ST\_RemoveIrrelevantPointsForView** - Availability: 3.5.0 Removes points that are irrelevant for rendering a specific rectangular view of a geometry.
- **ST\_RemoveSmallParts** - Availability: 3.5.0 Removes small parts (polygon rings or linestrings) of a geometry.
- **TopoGeo\_LoadGeometry** - Availability: 3.5.0 Load a geometry into an existing topology, snapping and splitting as needed.

## Functions enhanced in PostGIS 3.5

- **ST\_Clip** - Enhanced: 3.5.0 - touched argument added. Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.

## Functions changed in PostGIS 3.5

- **ST\_AsGeoJSON** - Changed: 3.5.0 allow specifying the column containing the feature id Return a geometry or feature in GeoJSON format.
- **ST\_DFullyWithin** - Changed: 3.5.0 : the logic behind the function now uses a test of containment within a buffer, rather than the ST\_MaxDistance algorithm. Results will differ from prior versions, but should be closer to user expectations. Tests if a geometry is entirely inside a distance of another

### 13.12.3 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.
- **PostGIS\_PROJ\_Compiled\_Version** - Availability: 3.5.0 Returns the version number of the PROJ library against which PostGIS was built.
- **RenameTopoGeometryColumn** - Availability: 3.4.0 Renames a topogeometry column
- **RenameTopology** - Availability: 3.4.0 Renames a topology
- **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 Window function that finds locations where polygons fail to form a valid coverage.
- **ST\_CoverageSimplify** - Availability: 3.4.0 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 extended forwards and backwards by specified distances.
- **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.
- **TopoElement** - Availability: 3.4.0 Converts a topogeometry to a topoelement.
- **debug\_standardize\_address** - Availability: 3.4.0 Returns a json formatted text listing the parse tokens and standardizations
- **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\_LineSubstring** - Enhanced: 3.4.0 - Support for geography was introduced. Returns the part of a line between two fractional locations.
- **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\_Resample** - Enhanced: 3.4.0 max and min resampling options added `ST_Resample(geometry, geometry, resample_option, max_size, min_size)`.
- **ST\_Rescale** - Enhanced: 3.4.0 max and min resampling options added 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\_ShortestLine** - Enhanced: 3.4.0 - support for geography. `ST_ShortestLine(geom1, geom2)`.

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.

### 13.12.4 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

- **RemoveUnusedPrimitives** - Availability: 3.3.0 Removes topology primitives which not needed to define existing TopoGeometry objects.
- **ST\_3DConvexHull** - Availability: 3.3.0 `ST_3DConvexHull(geometry)`.
- **ST\_3DUnion** - Availability: 3.3.0 aggregate variant was added Perform 3D union.
- **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** - Availability: 3.3.0 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** - Availability: 3.3.0 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.

### 13.12.5 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

- **FindLayer** - Availability: 3.2.0 Returns a topology.layer record by different means.
- **FindTopology** - Availability: 3.2.0 Returns a topology record by different means.
- **GetFaceContainingPoint** - Availability: 3.2.0 Finds the face containing a point.
- **ST\_AsFlatGeobuf** - Availability: 3.2.0 Return a FlatGeobuf representation of a set of rows.
- **ST\_Contour** - Availability: 3.2.0 Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.
- **ST\_DumpSegments** - Availability: 3.2.0
- **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\_InterpolateRaster** - Availability: 3.2.0 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\_SRID** - Availability: 3.2.0 Returns the spatial reference identifier for a topogeometry.
- **ST\_Scroll** - Availability: 3.2.0 Change start point of a closed LineString.
- **ST\_SetM** - Availability: 3.2.0 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\_SetZ** - Availability: 3.2.0 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.

- ## Functions enhanced in PostGIS 3.2

- ## Functions changed in PostGIS 3.2

- **ST\_Boundary** - Changed: 3.2.0 support for TIN, does not use geos, does not linearize curves `ST_Boundary(geometry)`.
- **ValidateTopology** - Changed: 3.2.0 added optional `bbox` parameter, perform face labeling and edge linking checks. Returns a set of `validate_topology` returntype objects detailing issues with topology.

### 13.12.6 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** - 2.1.0                                                                                                                                                   

## Functions enhanced in PostGIS 3.1

- **ST\_AsEWKT** - Enhanced: 3.1.0 support for optional precision parameter. `ST_AsEWKT(geometry, precision)` WKT(Well-Known Text) `SRID` `SRID`.
- **ST\_ClusterKMeans** - Enhanced: 3.1.0 Support for 3D geometries and weights Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST\_Difference** - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the part of geometry A that does not intersect geometry B.
- **ST\_Intersection** - Enhanced: 3.1.0 accept a gridSize parameter Computes a geometry representing the shared portion of geometries A and B.
- **ST\_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\_TileEnvelope** - `ST_TileEnvelope`: 2.0.0 `ST_TileEnvelope` `SRID` `SRID`. Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.
- **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\_Count** - 2.2.0 `ST_Count(rastertable, rastercolumn, ...)` Returns the number of non-null pixels in the specified column of the specified raster table. The function can also take a list of columns as input. If the column is not found, it returns null. The function can also take a value to exclude from the count. If the value is not found, it returns null. The function can also take a value to exclude from the count. If the value is not found, it returns null.

- **ST\_Force3D** - Changed: 3.1.0. Added support for supplying a non-zero Z value. `ST_Force3DZ(geometry, z_value)`. `ST_Force3DZ(geometry, z_value)`.
- **ST\_Force3DM** - Changed: 3.1.0. Added support for supplying a non-zero M value. `ST_Force3DM(geometry, m_value)`.
- **ST\_Force3DZ** - Changed: 3.1.0. Added support for supplying a non-zero Z value. `ST_Force3DZ(geometry, z_value)`.
- **ST\_Force4D** - Changed: 3.1.0. Added support for supplying non-zero Z and M values. `ST_Force4D(geometry, z_value, m_value)`.
- **ST\_Histogram** - Changed: 3.1.0 Removed `ST_Histogram(table_name, column_name)` variant. `ST_Histogram(bin_width, table_name, column_name)`.
- **ST\_Quantile** - Changed: 3.1.0 Removed `ST_Quantile(table_name, column_name)` variant. `ST_Quantile(population, quantile)`. `ST_Quantile(25%, 50%, 75%)` (percentile) `ST_Quantile(percentile)`.
- **ST\_SummaryStats** - 2.2.0 `ST_SummaryStats(rastertable, rastercolumn, ...)` 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 if no band is specified.

### 13.12.7 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

- **CG\_ConstrainedDelaunayTriangles** - 2.1.0 `CG_ConstrainedDelaunayTriangles(geometry)`. Return a constrained Delaunay triangulation around the given input geometry.
- **ST\_3DLineInterpolatePoint** - 2.1.0 `ST_3DLineInterpolatePoint(geometry, fraction)`. Returns a point interpolated along a 3D line at a fractional location.
- **ST\_ConstrainedDelaunayTriangles** - 2.1.0 `ST_ConstrainedDelaunayTriangles(geometry)`. Return a constrained Delaunay triangulation around the given input geometry.
- **ST\_TileEnvelope** - 2.1.0 `ST_TileEnvelope(srid, x, y, z)`. 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 a multipoint of random points contained in a Polygon or MultiPolygon.
- **ST\_GeomFromGeoJSON** - Enhanced: 3.0.0 parsed geometry defaults to SRID=4326 if not specified otherwise. GeoJSON to 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, GEOS version (SRS 3D) 3D distance.
- **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.
- **ST\_AsGeoJSON** - Changed: 3.0.0 support records as input Return a geometry or feature in GeoJSON format.
- **ST\_AsGeoJSON** - Changed: 3.0.0 output SRID if not EPSG:4326. Return a geometry or feature in GeoJSON format.
- **ST\_AsKML** - Changed: 3.0.0 - Removed the "versioned" variant signature GML 2 GML 3



- **ST\_Distance** - Changed: 3.0.0 - does not depend on SFCGAL anymore. 3 (longest) .
- **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.

### 13.12.8 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 3 (longest) .
- **ST\_AsHexWKB** - Availability: 2.5.0 Return the Well-Known Binary (WKB) in Hex representation of the raster.
- **ST\_BandFileSize** - Availability: 2.5.0 Returns the file size of a band stored in file system. If no bandnum specified, 1 is assumed.
- **ST\_BandFileTimestamp** - Availability: 2.5.0 Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.
- **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\_Grayscale** - Availability: 2.5.0 Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue
- **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
- **ST\_RastFromHexWKB** - Availability: 2.5.0 Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.
- **ST\_RastFromWKB** - Availability: 2.5.0 Return a raster value from a Well-Known Binary (WKB) raster.
- **ST\_SetBandIndex** - Availability: 2.5.0 Update the external band number of an out-db band
- **ST\_SetBandPath** - Availability: 2.5.0 Update the external path and band number of an out-db band

Functions enhanced in PostGIS 2.5

- **ST\_AsBinary/ST\_AsWKB** - Enhanced: 2.5.0 Addition of ST\_AsWKB Return the Well-Known Binary (WKB) representation of the raster.

- **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. `WKT(Well-Known Text) SRID`.
- **ST\_BandMetaData** - Enhanced: 2.5.0 to include `outdbbandnum`, `filesize` and `filetimestamp` for outdb rasters. `1`.
- **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. GeoJSON
- **ST\_GeometricMedian** - Enhanced: 2.5.0 Added support for `M` as weight of points. (median)
- **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.

Functions changed in PostGIS 2.5

- **ST\_GDALDrivers** - Changed: 2.5.0 - add `can_read` and `can_write` columns. Returns a list of raster formats supported by PostGIS through GDAL. Only those formats with `can_write=True` can be used by `ST_AsGDALRaster`

### 13.12.9 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** - 2.2.0. Return a Geobuf representation of a set of rows.
- **ST\_AsMVT** - 2.2.0. Aggregate function returning a MVT representation of a set of rows.
- **ST\_AsMVTGeom** - 2.2.0. Transforms a geometry into the coordinate space of a MVT tile.
- **ST\_Centroid** - Availability: 2.4.0 support for geography was introduced.
- **ST\_ForcePolygonCCW** - 2.2.0. Orients all exterior rings counter-clockwise and all interior rings clockwise.



- **ST\_ForcePolygonCW** - 2.2.0 関数。Orientes all exterior rings clockwise and all interior rings counter-clockwise.
- **ST\_FrechetDistance** - Availability: 2.4.0 - requires GEOS >= 3.7.0 関数 (shortest) 関数。
- **ST\_IsPolygonCCW** - 2.2.0 関数。Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.
- **ST\_IsPolygonCW** - 2.2.0 関数。Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.
- **ST\_MakeEmptyCoverage** - 2.2.0 関数。Cover georeferenced area with a grid of empty raster tiles.

#### Functions enhanced in PostGIS 2.4

- **Loader\_Generate\_Nation\_Script** - 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. 関数, 関数。
- **Normalize\_Address** - Enhanced: 2.4.0 norm\_addy object includes additional fields zip4 and address\_alphanumeric. 関数, 関数, 関数, 関数 norm\_addy 関数。関数 tiger\_geocoder 関数 (TIGER 関数) 関数。
- **Page\_Normalize\_Address** - Enhanced: 2.4.0 norm\_addy object includes additional fields zip4 and address\_alphanumeric. 関数, 関数, 関数, 関数 norm\_addy 関数。関数 tiger\_geocoder 関数 (TIGER 関数) 関数。address\_standardizer 関数。
- **Reverse\_Geocode** - 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 for details on loading zcta5 data. 関数。include\_strnum\_range = true 関数, 関数。
- **ST\_AsTWKB** - Enhanced: 2.4.0 memory and speed improvements. 関数 TWKB(Tiny Well-Known Binary) 関数。
- **ST\_Covers** - Enhanced: 2.4.0 Support for polygon in polygon and line in polygon added for geography type Tests if every point of B lies in A
- **ST\_CurveToLine** - Enhanced: 2.4.0 added support for max-deviation and max-angle tolerance, and for symmetric output. Converts a geometry containing curves to a linear geometry.
- **ST\_Project** - Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth. Returns a point projected from a start point by a distance and bearing (azimuth).
- **ST\_Reverse** - Enhanced: 2.4.0 support for curves was introduced. 関数 関数。

#### Functions changed in PostGIS 2.4

- **=** - Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use instead. Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **ST\_Node** - Changed: 2.4.0 this function uses GEOSNode internally instead of GEOSUnaryUnion. This may cause the resulting linestrings to have a different order and direction compared to PostGIS < 2.4. Nodes a collection of lines.

### 13.12.10 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).
- **Populate\_Topology\_Layer** - 2.3.0. Adds missing entries to topology.layer table by reading metadata from topo tables.
- **ST\_ClusterDBSCAN** - 2.3.0. Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- **ST\_ClusterKMeans** - 2.3.0. Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST\_GeneratePoints** - 2.3.0. Generates a multipoint of random points contained in a Polygon or MultiPolygon.
- **ST\_GeometricMedian** - 2.3.0. (median) .
- **ST\_MakeLine** - 2.0.0. , .
- **ST\_MinimumBoundingRadius** - 2.3.0. Returns the center point and radius of the smallest circle that contains a geometry.

- **ST\_MinimumClearance** - 2.3.0 `geometry geometry float`. `geometry` (robustness) `geometry` `float` (clearance) `float`.
- **ST\_MinimumClearanceLine** - 2.3.0 `geometry geometry geometry`. GEOS 3.6.0 `geometry`. `int` 2 `geometry`, `geometry`.
- **ST\_Normalize** - 2.3.0 `geometry geometry`. `geometry`.
- **ST\_Points** - 2.3.0 `geometry geometry`. `geometry`.
- **ST\_VoronoiLines** - 2.3.0 `geometry geometry`. Returns the boundaries of the Voronoi diagram of the vertices of a geometry.
- **ST\_VoronoiPolygons** - 2.3.0 `geometry geometry`. Returns the cells of the Voronoi diagram of the vertices of a geometry.
- **ST\_WrapX** - Availability: 2.3.0 requires GEOS X `geometry geometry`.
- **TopoGeom\_addElement** - 2.3 `topogeom geometry`. Adds an element to the definition of a TopoGeometry.
- **TopoGeom\_remElement** - 2.3 `topogeom geometry`. Removes an element from the definition of a TopoGeometry.
- **~(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** - `geometry`: 2.3.0 `geometry` (-1 `geometry`) `geometry`. `ST_LineString` `geometry` `ST_CircularString` `geometry`.

### 13.12.11 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

- <<->> - 2.2.0 `geometry`. PostgreSQL 9.1 `KNN geometry`. Returns the n-D distance between the A and B geometries or bounding boxes
- **ST\_3DDifference** - 2.2.0 `geometry`. 3 `geometry`.
- **ST\_3DUnion** - 2.2.0 `geometry`. Perform 3D union.
- **ST\_ApproximateMedialAxis** - 2.2.0 `geometry`. `geometry`.
- **ST\_AsEncodedPolyline** - 2.2.0 `text`. `geometry`
- **ST\_AsTWKB** - 2.2.0 `bytea`. `geometry` TWKB(Tiny Well-Known Binary) `bytea`.
- **ST\_BoundingDiagonal** - 2.2.0 `geometry`. `geometry`.
- **ST\_CPAWithin** - 2.2.0 `boolean`. Tests if the closest point of approach of two trajectories is within the specified distance.
- **ST\_ClipByBox2D** - 2.2.0 `geometry`. Computes the portion of a geometry falling within a rectangle.
- **ST\_ClosestPointOfApproach** - 2.2.0 `float8`. Returns a measure at the closest point of approach of two trajectories.
- **ST\_ClusterIntersecting** - 2.2.0 `setof geometry`. Aggregate function that clusters input geometries into connected sets.
- **ST\_ClusterWithin** - 2.2.0 `setof geometry`. Aggregate function that clusters geometries by separation distance.
- **ST\_CountAgg** - 2.2.0 `integer`. `geometry`. `integer`. `exclude_nodata_value` `boolean`, NODATA `boolean`.
- **ST\_CreateOverview** - 2.2.0 `geometry`. `geometry`.
- **ST\_DistanceCPA** - 2.2.0 `float8`. Returns the distance between the closest point of approach of two trajectories.
- **ST\_ForceCurve** - 2.2.0 `geometry`. `geometry`, `enum` (`upcast`) `enum`.
- **ST\_IsPlanar** - 2.2.0 `boolean`. `enum` 2.1.0 `enum` 2.1 `enum` `enum`. `enum`.
- **ST\_IsSolid** - 2.2.0 `boolean`. `enum`. `enum`.
- **ST\_IsValidTrajectory** - 2.2.0 `boolean`. Tests if the geometry is a valid trajectory.

- **ST\_LineFromEncodedPolyline** - 2.2.0 `geometry`. `geometry` (polyline) `geometry`.
- **ST\_MakeSolid** - 2.2.0 `geometry`. `geometry`. `geometry`.
- **ST\_MapAlgebra (callback function version)** - 2.2.0 `mask` `geometry`. `geometry` - `geometry` 1 `geometry`, `geometry`, `geometry` 1 `geometry` 1 `geometry`.
- **ST\_MemSize** - 2.2.0 `geometry`. `geometry` (`geometry`) `geometry`.
- **ST\_RemoveRepeatedPoints** - 2.2.0 `geometry`. Returns a version of a geometry with duplicate points removed.
- **ST\_Retile** - 2.2.0 `geometry`. `geometry`, `geometry`.
- **ST\_SetEffectiveArea** - 2.2.0 `geometry`. Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.
- **ST\_SimplifyVW** - 2.2.0 `geometry`. Returns a simplified representation of a geometry, using the Visvalingam-Whyatt algorithm
- **ST\_Subdivide** - 2.2.0 `geometry`. Computes a rectilinear subdivision of a geometry.
- **ST\_SummaryStatsAgg** - 2.2.0 `geometry`. 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 if no band is specified.
- **ST\_SwapOrdinates** - 2.2.0 `geometry`. `geometry`.
- **ST\_Volume** - 2.2.0 `geometry`. 3 `geometry`. `geometry` (`geometry`) 0 `geometry`.
- **parse\_address** - 2.2.0 `geometry`. `geometry`.
- **postgis.enable\_outdb\_rasters** - 2.2.0 `geometry`. DB `geometry`.
- **postgis.gdal\_datapath** - 2.2.0 `geometry`. GDAL `GDAL_DATA` `geometry`. `geometry`, `geometry` `GDAL_DATA` `geometry`.
- **postgis.gdal\_enabled\_drivers** - 2.2.0 `geometry`. PostGIS `geometry` GDAL `geometry`. GDAL `GDAL_SKIP` `geometry`.
- **standardize\_address** - 2.2.0 `geometry`. `geometry`, `geometry`, `geometry` `stdaddr` `geometry`.
- **|=** - 2.2.0 `geometry`. PostgreSQL 9.5 `geometry` (index-supported) `geometry`. A `B` `geometry` (closest point of approach) `geometry` (trajectory) `geometry`.

## Functions enhanced in PostGIS 2.2

- **<->** - **2.2.0** -- PostgreSQL 9.5 **KNN**("K nearest neighbor") **PostgreSQL 9.4** **KNN** **PostgreSQL 9.4** **A B 2** **PostgreSQL 9.4**.
- **AsTopoJSON** - **2.2.1** **(puntal)** **TopoGeometry** **TopoJSON**.
- **ST\_Area** - **2.2.0** **GeographicLib** **Proj 4.9.0**.

- **ST\_AsX3D** - 新增: 2.2.0 支持 X3D 编码 (x/y, z/height) 的 X3D 模型。支持 X3D XML 编码: ISO-IEC-19776-1.2-X3DEncodings-XML 模型。
- **ST\_Azimuth** - 新增: 2.2.0 支持 GeographicLib 2.2.0 版本。支持 Proj 4.9.0 版本。支持 2 个参数。
- **ST\_Distance** - 新增: 2.2.0 支持 GeographicLib 2.2.0 版本。支持 Proj 4.9.0 版本。支持 3 个参数 (longest) 模型。
- **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** - 新增: 2.2.0 支持 TIN 模型 (curve) 模型。支持 2 个参数。

### Functions changed in PostGIS 2.2

- **<->** - 新增: 2.2.0 支持 -- PostgreSQL 9.5 混合语法 (Hybrid syntax) 模型。支持 PostgreSQL 9.5 混合语法。A B 模型 2 个参数。
- **Get\_Geocode\_Setting** - 新增: 2.2.0 支持 geocode\_settings\_default 模型。支持 geocode\_settings 模型, 支持 geocode\_settings 模型。tiger.geocode\_settings 模型。
- **ST\_3DClosestPoint** - 新增: 2.2.0 支持 2D 模型, (支持 Z 0 模型) 2D 模型。2D 3D 模型, 支持 Z 模型 Z 0 模型。g2 模型 g1 模型 3 模型。支持 3D 模型。
- **ST\_3DDistance** - 新增: 2.2.0 支持, 2D 3D 模型 Z 模型 Z 0 模型。支持, 支持 (SRS 模型) 3 模型。
- **ST\_3DLongestLine** - 新增: 2.2.0 支持 2D 模型, (支持 Z 0 模型) 2D 模型。2D 3D 模型, 支持 Z 模型 Z 0 模型。支持 3 模型 (longest) 模型。
- **ST\_3DMaxDistance** - 新增: 2.2.0 支持, 2D 3D 模型 Z 模型 Z 0 模型。支持, 支持 (SRS 模型) 3 模型。
- **ST\_3DShortestLine** - 新增: 2.2.0 支持 2D 模型, (支持 Z 0 模型) 2D 模型。2D 3D 模型, 支持 Z 模型 Z 0 模型。支持 3 模型 (shortest) 模型。
- **ST\_DistanceSphere** - 新增: 2.2.0 支持 ST\_Distance\_Sphere 模型。支持 2 个参数。PostGIS 1.5 模型。
- **ST\_DistanceSpheroid** - 新增: 2.2.0 支持 ST\_Distance\_Spheroid 模型。支持 2 个参数。PostGIS 1.5 模型。
- **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** - 新增: 2.2.0 支持 ST\_Length\_Spheroid 模型, ST\_3DLength\_Spheroid 模型。支持 2 个参数。

- **ST\_MemSize** - Changed: 2.2.0 name changed to ST\_MemSize to follow naming convention. ST\_Geometry 返回内存大小。
- **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
- **ValidateTopology** - 新增: 2.2.0 检查'edge crosses node' 拓扑问题 id1 和 id2 返回 validate\_topology\_return\_type 对象 detailing issues with topology.

### 13.12.12 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

- **=** - 2.1.0 新增。A 是否等于 B 返回 TRUE 或 FALSE。
- **AsTopoJSON** - 2.1.0 新增。TopoGeometry 转 TopoJSON 格式。
- **Drop\_Nation\_Tables\_Generate\_Script** - 2.1.0 新增。生成 county\_all, state\_all 表结构, 生成 county, state 表 (州) 数据。
- **Get\_Geocode\_Setting** - 2.1.0 新增。tiger.geocode\_settings 返回设置。
- **Loader\_Generate\_Nation\_Script** - 2.1.0 新增。生成 nation 表结构, 生成 nation 表数据。
- **Page\_Normalize\_Address** - 2.1.0 新增。生成 norm\_addy 表, 生成 tiger\_geocoder 表 (TIGER 数据) 和 address\_standardizer 表。
- **ST\_3DArea** - 2.1.0 新增。3D 面积。
- **ST\_3DIntersection** - 2.1.0 新增。3D 交集。
- **ST\_Box2dFromGeoHash** - 2.1.0 新增。GeoHash 转 BOX2D 格式。
- **ST\_ColorMap** - 2.1.0 新增。生成 8BUI 表 (grayscale, RGB, RGBA) 和 4 个颜色表。
- **ST\_Contains** - 2.1.0 新增。检查 rastA 是否包含 rastB。
- **ST\_ContainsProperly** - 2.1.0 新增。检查 rastB 是否完全包含在 rastA 内部。
- **ST\_CoveredBy** - 2.1.0 新增。检查 rastA 是否被 rastB 覆盖。
- **ST\_Covers** - 2.1.0 新增。检查 rastB 是否覆盖 rastA。
- **ST\_DFullyWithin** - 2.1.0 新增。检查 rastA 是否完全在 rastB 内部。



- **ST\_DWithin** - 2.1.0 新增。返回 rastA 和 rastB 在指定距离内的像素。  
返回布尔值。
- **ST\_DelaunayTriangles** - 2.1.0 新增。Returns the Delaunay triangulation of the vertices of a geometry.
- **ST\_Disjoint** - 2.1.0 新增。返回 rastA 和 rastB 是否不相交。  
返回布尔值。
- **ST\_DumpValues** - 2.1.0 新增。返回栅格中所有像素值的数组。
- **ST\_Extrude** - 2.1.0 新增。将 2D 几何体沿 Z 轴拉伸。
- **ST\_ForceLHR** - 2.1.0 新增。LHR(Left Hand Reverse; 左手) 强制几何体。  
返回几何体。
- **ST\_FromGDALRaster** - 2.1.0 新增。从 GDAL 栅格创建几何体。  
返回几何体。
- **ST\_GeomFromGeoHash** - 2.1.0 新增。GeoHash 字符串转换为几何体。  
返回几何体。
- **ST\_InvDistWeight4ma** - 2.1.0 新增。4 邻域反距离权重。  
返回浮点值。
- **ST\_MapAlgebra (callback function version)** - 2.1.0 新增。使用回调函数进行栅格代数。  
返回栅格，输入为 1 个栅格，输出为 1 个栅格。
- **ST\_MapAlgebra (expression version)** - 2.1.0 新增。使用 SQL 表达式进行栅格代数。  
返回栅格，输入为 1 个栅格，输出为 1 个栅格。
- **ST\_MinConvexHull** - 2.1.0 新增。计算最小凸包。  
返回几何体。
- **ST\_MinDist4ma** - 2.1.0 新增。4 邻域最小距离。  
(返回浮点值) 返回浮点值。
- **ST\_MinkowskiSum** - 2.1.0 新增。闵可夫斯基和。  
返回几何体。
- **ST\_NearestValue** - 2.1.0 新增。columnx 和 rowy 指定位置的最近值。  
返回浮点值，NODATA 返回 NULL。
- **ST\_Neighborhood** - 2.1.0 新增。columnx 和 rowy 指定位置的邻域。  
返回栅格，NODATA 返回 NULL。
- **ST\_NotSameAlignmentReason** - 2.1.0 新增。返回不同对齐的原因。  
返回文本。
- **ST\_Orientation** - 2.1.0 新增。返回几何体的方位。  
返回文本 (orientation)。
- **ST\_Overlaps** - 2.1.0 新增。返回 rastA 和 rastB 是否重叠。  
返回布尔值。
- **ST\_PixelAsCentroid** - 2.1.0 新增。返回指定像素的质心。  
返回几何体。
- **ST\_PixelAsCentroids** - 2.1.0 新增。返回所有像素的质心。  
(返回几何体) 返回几何体 X, Y 坐标。
- **ST\_PixelAsPoint** - 2.1.0 新增。返回指定像素的点。  
返回几何体。
- **ST\_PixelAsPoints** - 2.1.0 新增。返回所有像素的点。  
返回几何体 X, Y 坐标。
- **ST\_PixelOfValue** - 2.1.0 新增。返回指定值的像素。  
返回几何体。



- **ST\_PointFromGeoHash** - 2.1.0 从 GeoHash 字符串生成点。
- **ST\_RasterToWorldCoord** - 2.1.0 从栅格坐标 (X, Y) 返回世界坐标。返回 1 个元组。
- **ST\_Resize** - 2.1.0 使用 GDAL 1.6.1 的 `GDAL_Resize` 函数调整栅格大小。
- **ST\_Roughness** - 2.1.0 计算 DEM 的粗糙度 (roughness)。
- **ST\_SetValues** - 2.1.0 使用元组列表更新栅格值。
- **ST\_Simplify** - 2.1.0 使用 Douglas-Peucker 算法简化 TopoGeometry。
- **ST\_StraightSkeleton** - 2.1.0 计算直骨架 (straight skeleton)。
- **ST\_Summary** - 2.1.0 返回栅格的元数据摘要。
- **ST\_TPI** - 2.1.0 计算地形位置指数 (Topographic Position Index)。
- **ST\_TRI** - 2.1.0 计算地形粗糙度指数 (Terrain Ruggedness Index)。
- **ST\_Tessellate** - 2.1.0 对多边形进行三角剖分 (tessellation) 生成 TIN。
- **ST\_Tile** - 2.1.0 将栅格切片成指定大小的瓦片。
- **ST\_Touches** - 2.1.0 检查两个栅格是否接触 (touches)。
- **ST\_Union** - 2.1.0 使用 `ST_Union(rast, unionarg)` 合并栅格。返回 1 个元组。
- **ST\_Within** - 2.1.0 检查栅格 B 是否完全在栅格 A 内部。
- **ST\_WorldToRasterCoord** - 2.1.0 从世界坐标 X, Y 返回栅格坐标。
- **Set\_Geocode\_Setting** - 2.1.0 设置地理编码相关的配置。
- **UpdateRasterSRID** - 2.1.0 更新栅格的 SRID 值。
- **clearTopoGeom** - 2.1 清除拓扑几何体的内容。
- **postgis\_sfcgal\_version** - 2.1.0 返回 SFCGAL 的版本信息。

#### Functions enhanced in PostGIS 2.1

- **ST\_AddBand** - 2.1.0 使用 `addbandarg` 参数添加新的波段。

- **ST\_AddBand** - 新增: 2.1.0 支持在 DB 中存储栅格数据。支持在栅格数据中添加新的波段 (band) 或波段名称。支持在栅格数据中添加新的波段名称, 支持在栅格数据中添加新的波段名称。
- **ST\_AsBinary/ST\_AsWKB** - 新增: 2.1.0 支持 outasbin 函数。Return the Well-Known Binary (WKB) representation of the raster.
- **ST\_AsGML** - 新增: 2.1.0 支持 GML 3 的 ID 属性。支持 GML 2 和 GML 3 的 ID 属性。
- **ST\_Aspect** - 新增: 2.1.0 支持 ST\_MapAlgebra() 函数, 支持 interpolate\_nodata 函数。支持 ST\_MapAlgebra() 函数 (支持 interpolate\_nodata) 函数。支持 ST\_MapAlgebra() 函数。
- **ST\_Boundary** - 新增: 2.1.0 支持在栅格数据中提取边界。支持在栅格数据中提取边界。
- **ST\_Clip** - 新增: 2.1.0 支持 C 语言函数。Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.
- **ST\_DWithin** - 增强: 2.1.0 improved speed for geography. See Making Geography faster for details. Tests if two geometries are within a given distance
- **ST\_DWithin** - 增强: 2.1.0 support for curved geometries was introduced. Tests if two geometries are within a given distance
- **ST\_Distance** - 增强: 2.1.0 improved speed for geography. See Making Geography faster for details. 支持 3 个参数 (longest) 的 ST\_Distance 函数。
- **ST\_Distance** - 新增: 2.1.0 支持在栅格数据中提取距离。支持 3 个参数 (longest) 的 ST\_Distance 函数。
- **ST\_Distinct4ma** - 新增: 2.1.0 支持在栅格数据中提取 2 个参数。支持在栅格数据中提取 2 个参数。
- **ST\_DumpPoints** - 增强: 2.1.0 Faster speed. Reimplemented as native-C. 支持在栅格数据中提取点。
- **ST\_HillShade** - 新增: 2.1.0 支持 ST\_MapAlgebra() 函数, 支持 interpolate\_nodata 函数。支持 ST\_MapAlgebra() 函数, 支持 interpolate\_nodata 函数。支持 ST\_MapAlgebra() 函数。
- **ST\_MakeValid** - 增强: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT. Attempts to make an invalid geometry valid without losing vertices.
- **ST\_Max4ma** - 新增: 2.1.0 支持在栅格数据中提取 2 个参数。支持在栅格数据中提取 2 个参数。
- **ST\_Mean4ma** - 新增: 2.1.0 支持在栅格数据中提取 2 个参数。支持在栅格数据中提取 2 个参数。
- **ST\_Min4ma** - 新增: 2.1.0 支持在栅格数据中提取 2 个参数。支持在栅格数据中提取 2 个参数。
- **ST\_PixelAsPolygons** - 新增: 2.1.0 支持 exclude\_nodata\_value 函数。支持 ST\_PixelAsPolygons 函数 X, Y 属性。
- **ST\_Polygon** - 新增: 2.1.0 支持在栅格数据中提取多边形 (支持 C 语言函数)。支持在栅格数据中提取多边形 (支持 C 语言函数)。NODATA 支持在栅格数据中提取多边形 (支持 C 语言函数)。
- **ST\_Range4ma** - 新增: 2.1.0 支持在栅格数据中提取 2 个参数。支持在栅格数据中提取 2 个参数。

- **ST\_SameAlignment** - 新增: 2.1.0 检查两个几何体是否具有相同的对齐方式。如果两个几何体的对齐方式相同，则返回 true，否则返回 false。 (检查两个几何体的对齐方式是否相同，如果相同则返回 true，否则返回 false)。
- **ST\_Segmentize** - 新增: 2.1.0 将几何体分割成指定的最大段长。 Returns a modified geometry/geography having no segment longer than a given distance.
- **ST\_SetGeoReference** - 新增: 2.1.0 设置 ST\_SetGeoReference(raster, double precision, ...) 的地理参考信息。最多支持 6 个参数。GDAL 的 ESRI 投影信息。GDAL 的。
- **ST\_SetValue** - 新增: 2.1.0 设置 ST\_SetValue() 的几何体值。 ST\_SetValues() 的 geomval[] 的 wrapper 函数。 columnx, rowy 的几何体值。 1 的几何体值， 1 的几何体值。
- **ST\_Slope** - 新增: 2.1.0 设置 ST\_MapAlgebra() 的 units, scale, interpolate nodata 的几何体值。 (检查两个几何体的对齐方式是否相同，如果相同则返回 true，否则返回 false)。
- **ST\_StdDev4ma** - 新增: 2.1.0 设置 2 的几何体值。 2 的几何体值。
- **ST\_Sum4ma** - 新增: 2.1.0 设置 2 的几何体值。 2 的几何体值。
- **ST\_Summary** - 新增: 2.1.0 设置 S 的几何体值。 S 的几何体值。
- **ST\_Transform** - 新增: 2.1.0 设置 ST\_Transform(rast, alignto) 的几何体值。 NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos 的几何体值。 NearestNeighbor 的几何体值。
- **ST\_Union** - 新增: 2.1.0 设置 (检查 C 的几何体值)。 1 的几何体值。
- **ST\_Union** - 新增: 2.1.0 设置 ST\_Union(rast) 的 1 的几何体值。 PostGIS 的 1 的几何体值。
- **ST\_Union** - 新增: 2.1.0 设置 ST\_Union(rast, uniontype) 的 4 的几何体值。 1 的几何体值。
- **toTopoGeom** - 新增: 2.1.0 设置 TopoGeometry 的几何体值。 Converts a simple Geometry into a topo geometry.

#### Functions changed in PostGIS 2.1

- **ST\_Aspect** - 新增: 2.1.0 设置 2.1.0 的几何体值。 2.1.0 的几何体值。
- **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** - 新增: 2.1.0 设置, 2.0.x 设置 ST\_Force\_2D 的几何体值。 "2" 的几何体值。
- **ST\_Force3D** - 新增: 2.1.0 设置, 2.0.x 设置 ST\_Force\_3D 的 XYZ 的几何体值。 ST\_Force3DZ 的几何体值。
- **ST\_Force3DM** - 新增: 2.1.0 设置, 2.0.x 设置 ST\_Force\_3DM 的 XYM 的几何体值。

- **ST\_Force3DZ** - 函数: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_Force\_3DZ 实现。将 XYZ 坐标强制转换为 3D。
- **ST\_Force4D** - 函数: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_Force\_4D 实现。将 XYZM 坐标强制转换为 4D。
- **ST\_ForceCollection** - 函数: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_Force\_Collection 实现。将集合类型的几何体强制转换为集合类型。
- **ST\_HillShade** - 函数: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_HillShade 实现。2.1.0 版本中增加了 hillshade 参数, 用于指定阴影的方向和强度。默认值为 45 度, 强度为 1.0。
- **ST\_LineInterpolatePoint** - 函数: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_Line\_Interpolate\_Point 实现。Returns a point interpolated along a line at a fractional location.
- **ST\_LineLocatePoint** - 函数: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_Line\_Locate\_Point 实现。Returns the fractional location of the closest point on a line to a point.
- **ST\_LineSubstring** - 函数: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_Line\_Substring 实现。Returns the part of a line between two fractional locations.
- **ST\_PixelAsCentroids** - 函数: 2.1.1 新增, 在 2.0.x 版本中通过 ST\_PixelAsCentroids 实现。exclude\_nodata\_value 参数用于指定是否排除无数据值。默认值为 true。返回的几何体类型为 POINT (或 POINT\_Z) 类型。X, Y 坐标为浮点型。
- **ST\_PixelAsPoints** - 函数: 2.1.1 新增, 在 2.0.x 版本中通过 ST\_PixelAsPoints 实现。exclude\_nodata\_value 参数用于指定是否排除无数据值。默认值为 true。返回的几何体类型为 POINT (或 POINT\_Z) 类型。X, Y 坐标为浮点型。
- **ST\_PixelAsPolygons** - 函数: 2.1.1 新增, 在 2.0.x 版本中通过 ST\_PixelAsPolygons 实现。exclude\_nodata\_value 参数用于指定是否排除无数据值。默认值为 true。返回的几何体类型为 POLYGON (或 POLYGON\_Z) 类型。X, Y 坐标为浮点型。
- **ST\_Polygon** - 函数: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_Polygon 实现。NODATA 值将返回 NULL。
- **ST\_RasterToWorldCoordX** - 函数: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_Raster2WorldCoordX 实现。返回世界坐标 X。像素索引从 1 开始。
- **ST\_RasterToWorldCoordY** - 函数: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_Raster2WorldCoordY 实现。返回世界坐标 Y。像素索引从 1 开始。
- **ST\_Rescale** - 函数: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_Rescale 实现。SRID 参数用于指定新的 SRID。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\_Reskew** - 函数: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_Reskew 实现。SRID 参数用于指定新的 SRID。NearestNeighbor (或 NearestNeighbor), Bilinear, Cubic, CubicSpline 或 Lanczos 重采样算法。默认值为 NearestNeighbor。
- **ST\_Segmentize** - Changed: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_Segmentize 实现。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.
- **ST\_Slope** - 函数: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_Slope 实现。2.1.0 版本中增加了 hillshade 参数, 用于指定阴影的方向和强度。默认值为 45 度, 强度为 1.0。
- **ST\_SnapToGrid** - 函数: 2.1.0 新增, 在 2.0.x 版本中通过 ST\_SnapToGrid 实现。SRID 参数用于指定新的 SRID。NearestNeighbor (或 NearestNeighbor), Bilinear, Cubic, CubicSpline 或 Lanczos 重采样算法。默认值为 NearestNeighbor。

- **ST\_WorldToRasterCoordX** - 2.1.0 返回 ST\_World2RasterCoordX 的 X 坐标 (pt) 的 X 坐标 (xw, yw) 的 X 坐标。
- **ST\_WorldToRasterCoordY** - 2.1.0 返回 ST\_World2RasterCoordY 的 Y 坐标 (pt) 的 Y 坐标 (xw, yw) 的 Y 坐标。

### 13.12.13 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

- **&&** - 2.0.0 返回 A 和 B 是否相等。TRUE 表示相等。
- **&&&** - 2.0.0 返回 A 和 B 是否相等。TRUE 表示相等。
- **<#>** - 2.0.0 返回 PostgreSQL 9.1 的 KNN 索引。A 和 B 的 2 个索引。
- **<->** - 2.0.0 返回 KNN 索引。A 和 B 的 2 个索引。PostgreSQL 9.1 的 KNN 索引。
- **@** - 2.0.0 返回 raster @ raster, raster @ geometry 是否相等。B 是否等于 A 的 TRUE 表示相等。
- **@** - 2.0.5 返回 geometry @ raster 是否相等。B 是否等于 A 的 TRUE 表示相等。
- **AddEdge** - 2.0.0 返回 ID(edgeid) 的边。返回 ID(edgeid) 的边。
- **AddFace** - 2.0.0 返回 ID(faceid) 的面。返回 ID(faceid) 的面。
- **AddNode** - 2.0.0 返回 ID(nodeid) 的节点。返回 ID(nodeid) 的节点。
- **AddOverviewConstraints** - 2.0.0 返回 (overview) 的约束。返回 (overview) 的约束。
- **AddRasterConstraints** - 2.0.0 返回 raster constraints 的约束。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.
- **AsGML** - 2.0.0 返回 TopoGeometry 的 GML 格式。
- **CopyTopology** - 2.0.0 返回 topology (nodes, edges, faces, layers and TopoGeometries) 的副本。Makes a copy of a topology (nodes, edges, faces, layers and TopoGeometries) into a new schema
- **DropOverviewConstraints** - 2.0.0 返回 (overview) 的约束。返回 (overview) 的约束。
- **DropRasterConstraints** - 2.0.0 返回 raster constraints 的约束。Returns true if the constraint setting was accomplished and issues a notice otherwise.

- **Drop Indexes Generate Script** - 2.0.0 生成脚本。TIGER 数据表 (tiger\_data) 的索引。生成脚本 tiger\_data。
- **Drop State Tables Generate Script** - 2.0.0 生成脚本。生成脚本 (州) 的 tiger\_data。
- **Geocode Intersection** - 2.0.0 生成脚本。生成脚本 2 个, 生成脚本 NAD83 生成脚本 geomout, 生成脚本 normalized\_address (addy) 生成脚本, 生成脚本。生成脚本 (约 10) 生成脚本。TIGER 生成脚本 (edge, face, addr) 生成脚本 PostgreSQL 生成脚本 (soundex, levenshtein) 生成脚本。
- **GetEdgeByPoint** - 2.0.0 生成脚本。 Finds the edge-id of an edge that intersects a given point.
- **GetFaceByPoint** - 2.0.0 生成脚本。 Finds face intersecting a given point.
- **GetNodeByPoint** - 2.0.0 生成脚本。 Finds the node-id of a node at a point location.
- **GetNodeEdges** - 2.0 生成脚本。生成脚本。
- **GetRingEdges** - 2.0.0 生成脚本。生成脚本。
- **GetTopoGeomElements** - 2.0.0 生成脚本。 Returns a set of topoelement objects containing the topological element\_id, element\_type of the given TopoGeometry (primitive elements).
- **GetTopologySRID** - 2.0.0 生成脚本。生成脚本 topology.topology 生成脚本 SRID 生成脚本。
- **Get Tract** - 2.0.0 生成脚本。生成脚本 (tract) 生成脚本 (field) 生成脚本。生成脚本。
- **Install Missing Indexes** - 2.0.0 生成脚本。生成脚本 (join) 生成脚本 (key) 生成脚本。
- **Loader Generate Census Script** - 2.0.0 生成脚本。生成脚本 (州) 生成脚本, TIGER 生成脚本 (州) 生成脚本 (tract), 生成脚本 (bg), 生成脚本 (tabblock) 生成脚本 tiger\_data 生成脚本。生成脚本 (州) 生成脚本。
- **Loader Generate Script** - 2.0.0 生成脚本。TIGER 2010 生成脚本 (tract), 生成脚本 (bg), 生成脚本 (tabblock) 生成脚本。生成脚本 (州) 生成脚本, TIGER 生成脚本 tiger\_data 生成脚本。生成脚本 (州) 生成脚本。生成脚本 TIGER 2010 生成脚本, 生成脚本, 生成脚本, 生成脚本。
- **Missing Indexes Generate Script** - 2.0.0 生成脚本。生成脚本 (join) 生成脚本 (key) 生成脚本 SQL DDL 生成脚本。
- **Polygonize** - 2.0.0 生成脚本。 Finds and registers all faces defined by topology edges.
- **Reverse Geocode** - 2.0.0 生成脚本。生成脚本。 include\_strnum\_range = true 生成脚本, 生成脚本。
- **ST\_3DClosestPoint** - 2.0.0 生成脚本。 g2 生成脚本 g1 生成脚本 3 生成脚本。生成脚本 3D 生成脚本。

- [illegible]

- [illegible]



- **ST\_MapAlgebraExpr** - 2.0.0 `geometry geometry integer`. Returns 1 if: PostgreSQL `geometry`, `geometry`, `int` 1 `geometry`. `geometry`, `int` 1 `geometry`.
- **ST\_MapAlgebraExpr** - 2.0.0 `geometry geometry integer`. Returns 2 if: PostgreSQL `geometry`, `geometry`, `int` 1 `geometry`. `geometry`, `int` 1 `geometry`. `extentype` `INTERSECTION`, `UNION`, `FIRST`, `SECOND` `geometry`.
- **ST\_MapAlgebraFct** - 2.0.0 `geometry geometry integer`. Returns 1 if: PostgreSQL `geometry`, `geometry`, `int` 1 `geometry`. `geometry`, `int` 1 `geometry`.
- **ST\_MapAlgebraFct** - 2.0.0 `geometry geometry integer`. Returns 2 if: PostgreSQL `geometry`, `geometry`, `int` 1 `geometry`. `geometry`, `int` 1 `geometry`. `INTERSECTION` `geometry`.
- **ST\_MapAlgebraFctNgb** - 2.0.0 `geometry geometry integer`. Returns PostgreSQL `geometry` (Map Algebra Nearest Neighbor) `neighborhood` PostgreSQL `geometry`.
- **ST\_Max4ma** - 2.0.0 `geometry geometry`. `geometry`.
- **ST\_Mean4ma** - 2.0.0 `geometry geometry`. `geometry`.
- **ST\_Min4ma** - 2.0.0 `geometry geometry`. `geometry`.
- **ST\_ModEdgeHeal** - 2.0 `geometry`. Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node.
- **ST\_MoveIsoNode** - 2.0.0 `geometry geometry`. Moves an isolated node in a topology from one point to another. If new apoint geometry exists as a node an error is thrown. Returns description of move.
- **ST\_NewEdgeHeal** - 2.0 `geometry geometry`. Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided.
- **ST\_Node** - 2.0.0 `geometry`. Nodes a collection of lines.
- **ST\_NumPatches** - 2.0.0 `geometry`. `geometry`. `NULL` `geometry`.
- **ST\_OffsetCurve** - 2.0 `geometry`. Returns an offset line at a given distance and side from an input line.
- **ST\_PatchN** - 2.0.0 `geometry`. ST\_Geometry `geometry`.
- **ST\_Perimeter** - `geometry`: 2.0.0 `geometry`. Returns the length of the boundary of a polygonal geometry or geography.
- **ST\_PixelAsPolygon** - 2.0.0 `geometry`. `geometry`.
- **ST\_PixelAsPolygons** - 2.0.0 `geometry`. `geometry` X, Y `geometry`.
- **ST\_Project** - 2.0.0 `geometry`. Returns a point projected from a start point by a distance and bearing (azimuth).

- **ST\_Quantile** - 2.0.0 函数。将输入的多边形（population）按指定的分位数（quantile）进行划分。例如，将多边形划分为 25%、50%、75% 的分位数（percentile）。
- **ST\_Range4ma** - 2.0.0 函数。返回输入的多边形的 4 个最近邻的多边形。
- **ST\_Reclass** - 2.0.0 函数。将输入的多边形的值重新分类。nband 参数指定了输入的多边形的波段数。nband 参数指定了输入的多边形的波段数。例如：将 16BUI 重新分类为 8BUI。
- **ST\_RelateMatch** - 2.0.0 函数。Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern
- **ST\_RemEdgeModFace** - 2.0 函数。Removes an edge, and if the edge separates two faces deletes one face and modifies the other face to cover the space of both.
- **ST\_RemEdgeNewFace** - 2.0 函数。Removes an edge, and if the edge separates two faces deletes one face and creates a new face to cover the space of both.
- **ST\_Resample** - 2.0.0 函数。GDAL 1.6.1 函数。将输入的多边形重新采样为指定的分辨率。例如，将 100m 分辨率的多边形重新采样为 50m 分辨率。
- **ST\_Rescale** - 2.0.0 函数。GDAL 1.6.1 函数。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\_Reskew** - 2.0.0 函数。GDAL 1.6.1 函数。将输入的多边形的值重新分类。例如，将 100m 分辨率的多边形重新分类为 50m 分辨率。
- **ST\_SameAlignment** - 2.0.0 函数。检查两个多边形是否具有相同的对齐方式。例如，检查两个多边形是否具有相同的对齐方式。
- **ST\_SetBandIsNoData** - 2.0.0 函数。将输入的多边形的指定波段设置为 NoData。
- **ST\_SharedPaths** - 2.0.0 函数。返回两个多边形的共享路径。
- **ST\_Slope** - 2.0.0 函数。计算输入的多边形的坡度（或斜率）。
- **ST\_Snap** - 2.0.0 函数。将输入的多边形的顶点移动到指定的距离。
- **ST\_SnapToGrid** - 2.0.0 函数。GDAL 1.6.1 函数。将输入的多边形的顶点移动到指定的网格上。例如，将 100m 分辨率的多边形移动到 50m 分辨率的网格上。
- **ST\_Split** - Availability: 2.0.0 requires GEOS Returns a collection of geometries created by splitting a geometry by another geometry.
- **ST\_StdDev4ma** - 2.0.0 函数。返回输入的多边形的 4 个最近邻的多边形的标准差。
- **ST\_Sum4ma** - 2.0.0 函数。返回输入的多边形的 4 个最近邻的多边形的和。

- **ST\_SummaryStats** - 2.0.0 関数。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 if no band is specified.
- **ST\_Transform** - 2.0.0 関数。GDAL 1.6.1 関数。NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos 関数。NearestNeighbor 関数。
- **ST\_UnaryUnion** - 2.0.0 関数。Computes the union of the components of a single geometry.
- **ST\_Union** - 2.0.0 関数。1 関数。
- **ST\_ValueCount** - 2.0.0 関数。 ( ) 関数。 1 関数。 NODATA 関数。
- **TopoElementArray\_Agg** - 2.0.0 関数。Returns a topoelementarray for a set of element\_id, type arrays (topoelements).
- **TopoGeo\_AddLineString** - 2.0.0 関数。Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.
- **TopoGeo\_AddPoint** - 2.0.0 関数。 (split) 関数。
- **TopoGeo\_AddPolygon** - 2.0.0 関数。Adds a polygon to an existing topology using a tolerance and possibly splitting existing edges/faces. Returns face identifiers.
- **TopologySummary** - 2.0.0 関数。Takes a topology name and provides summary totals of types of objects in topology.
- **Topology\_Load\_Tiger** - 2.0.0 関数。PostGIS TIGER 関数。
- **toTopoGeom** - 2.0 関数。Converts a simple Geometry into a topo geometry.
- **~** - 2.0.0 関数。A 関数 B 関数 TRUE 関数。
- **~=** - 2.0.0 関数。A 関数 B 関数 TRUE 関数。

#### Functions enhanced in PostGIS 2.0

- **&&** - 2.0.0 関数 (polyhedral surface) 関数。A 2D 関数 B 2D 関数 TRUE 関数。
- **AddGeometryColumn** - 2.0.0 関数。use typmod 関数。 typmod 関数。
- **Box2D** - 2.0.0 関数, TIN 関数。Returns a BOX2D representing the 2D extent of a geometry.
- **Box3D** - 2.0.0 関数, TIN 関数。Returns a BOX3D representing the 3D extent of a geometry.
- **CreateTopology** - Enhanced: 2.0 added the signature accepting hasZ Creates a new topology schema and registers it in the topology.topology table.

- **Geocode** - 函数: 2.0.0 使用 TIGER 2010 数据, 返回给定地址的几何。使用 `max_results` 参数限制返回结果的数量。默认使用 NAD83 坐标系。使用 `restrict_region(NULL)` 限制结果范围。
- **GeometryType** - 函数: 2.0.0 返回几何的 SRID。ST\_Geometry 函数返回几何的 SRID。
- **Populate\_Geometry\_Columns** - 函数: 2.0.0 使用 `use_typmod` 参数。Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- **ST\_3DExtent** - 函数: 2.0.0 返回几何的 3D 边界框。Aggregate function that returns the 3D bounding box of geometries.
- **ST\_Affine** - 函数: 2.0.0 应用 3D 仿射变换到几何。
- **ST\_Area** - 函数: 2.0.0 返回 2 维多面体 (polyhedral surface) 的面积。
- **ST\_AsBinary** - 函数: 2.0.0 返回几何的 OGC/ISO Well-Known Binary (WKB) 表示, 不带 SRID 元数据。
- **ST\_AsBinary** - 函数: 2.0.0 返回几何的 OGC/ISO Well-Known Binary (WKB) 表示, 不带 SRID 元数据。
- **ST\_AsBinary** - 函数: 2.0.0 返回几何的 OGC/ISO Well-Known Binary (WKB) 表示, 不带 SRID 元数据。
- **ST\_AsEWKB** - 函数: 2.0.0 返回几何的 Extended Well-Known Binary (EWKB) 表示, 带 SRID 元数据。
- **ST\_AsEWKT** - 函数: 2.0.0 返回几何的 Extended Well-Known Text (EWKT) 表示, 带 SRID 元数据。
- **ST\_AsGML** - 函数: 2.0.0 返回几何的 GML 3.2.1 表示。GML 3.2.1 支持 TIN 几何。GML 2 和 GML 3 表示。
- **ST\_AsKML** - 函数: 2.0.0 返回几何的 KML 表示。GML 2 和 GML 3 表示。
- **ST\_Azimuth** - 函数: 2.0.0 返回从一点到另一点的方位角。
- **ST\_Dimension** - 函数: 2.0.0 返回几何的维度 (polyhedral surface) 的 TIN 表示。ST\_Geometry 函数返回几何的 SRID。
- **ST\_Dump** - 函数: 2.0.0 返回几何的组件。Returns a set of geometry\_dump rows for the components of a geometry.
- **ST\_DumpPoints** - 函数: 2.0.0 返回几何的点的集合。Returns a set of geometry\_dump rows for the components of a geometry.
- **ST\_Expand** - 函数: 2.0.0 返回从另一个边界框或几何扩展的边界框。
- **ST\_Extent** - 函数: 2.0.0 返回几何的边界框。Aggregate function that returns the bounding box of geometries.

- **ST\_Force2D** - 函数: 2.0.0 强制将多面体表面 (polyhedral surface) 转换为 2D。返回 "2D 多面体表面"。
- **ST\_Force3D** - 函数: 2.0.0 强制将多面体表面 (polyhedral surface) 转换为 3D。返回 XYZ 多面体表面。ST\_Force3DZ 同义函数。
- **ST\_Force3DZ** - 函数: 2.0.0 强制将多面体表面 (polyhedral surface) 转换为 3D。返回 XYZ 多面体表面。
- **ST\_ForceCollection** - 函数: 2.0.0 强制将多面体表面 (polyhedral surface) 转换为集合。返回多面体表面集合。
- **ST\_ForceRHR** - 函数: 2.0.0 强制将多面体表面 (polyhedral surface) 转换为右手规则 (orientation) 多面体表面 (Right-Hand Rule) 多面体表面。
- **ST\_GMLToSQL** - 函数: 2.0.0 将多面体表面 (polyhedral surface) 转换为 TIN 多面体表面。GML 多面体表面 ST\_Geometry 转换为 ST\_GeomFromGML 多面体表面。
- **ST\_GMLToSQL** - 函数: 2.0.0 将多面体表面 (polyhedral surface) 转换为 SRID 多面体表面。GML 多面体表面 ST\_Geometry 转换为 ST\_GeomFromGML 多面体表面。
- **ST\_GeomFromEWKB** - 函数: 2.0.0 将多面体表面 (polyhedral surface) 转换为 TIN 多面体表面。EWKB(Extended Well-Known Binary) 多面体表面 ST\_Geometry 转换为 ST\_GeomFromEWKB 多面体表面。
- **ST\_GeomFromEWKT** - 函数: 2.0.0 将多面体表面 (polyhedral surface) 转换为 TIN 多面体表面。EWKT(Extended Well-Known Text) 多面体表面 ST\_Geometry 转换为 ST\_GeomFromEWKT 多面体表面。
- **ST\_GeomFromGML** - 函数: 2.0.0 将多面体表面 (polyhedral surface) 转换为 TIN 多面体表面。从 GML 多面体表面转换为 PostGIS 多面体表面。
- **ST\_GeomFromGML** - 函数: 2.0.0 将多面体表面 (polyhedral surface) 转换为 SRID 多面体表面。从 GML 多面体表面转换为 PostGIS 多面体表面。
- **ST\_GeometryN** - 函数: 2.0.0 返回多面体表面 (polyhedral surface) 的第 N 个多面体表面。ST\_Geometry 返回多面体表面。
- **ST\_GeometryType** - 函数: 2.0.0 返回多面体表面 (polyhedral surface) 的多面体表面。ST\_Geometry 返回多面体表面。
- **ST\_IsClosed** - 函数: 2.0.0 检查多面体表面 (polyhedral surface) 是否闭合。LINESTRING 多面体表面返回 TRUE。多面体表面 (多面体表面) 返回 TRUE 或 FALSE。
- **ST\_MakeEnvelope** - 函数: 2.0 返回 SRID 多面体表面 (envelope) 多面体表面。返回 SRID 多面体表面 SRS 多面体表面。
- **ST\_MakeValid** - Enhanced: 2.0.1, speed improvements Attempts to make an invalid geometry valid without losing vertices.
- **ST\_NPoints** - 函数: 2.0.0 返回多面体表面 (polyhedral surface) 的多面体表面。返回多面体表面 (多面体表面) 多面体表面。
- **ST\_NumGeometries** - 函数: 2.0.0 返回多面体表面 (polyhedral surface) 的多面体表面。返回多面体表面 (多面体表面) 多面体表面。
- **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** - 函数: 2.0.0 返回多面体表面 (polyhedral surface) 的多面体表面。Rotates a geometry about an origin point.



- **Populate\_Geometry\_Columns** - 新增: 2.0.0 新增。强制所有几何列都使用 type modifiers 或具有适当的空间约束。
- **ST\_3DExtent** - 更改: 2.0.0 在以前的版本中这曾被称为 ST\_Extent3D Aggregate 函数，它返回几何体的 3D 包围盒。
- **ST\_3DLength** - 新增: 2.0.0 新增 ST\_Length3D 函数。返回几何体的 3D 长度。
- **ST\_3DMakeBox** - 更改: 2.0.0 在以前的版本中这曾被称为 ST\_MakeBox3D 函数，它创建由两个 3D 点几何体定义的 BOX3D。
- **ST\_3DPerimeter** - 新增: 2.0.0 新增 ST\_Perimeter3D 函数。返回几何体的 3D 周长。
- **ST\_AsBinary** - 新增: 2.0.0 新增 ST\_AsBinary(geometry) 函数。返回 OGC/ISO Well-Known Binary (WKB) 表示的几何体/地理数据，不带 SRID 元数据。
- **ST\_AsGML** - 新增: 2.0.0 新增 ST\_AsGML(geometry) 函数。返回 GML 2 或 GML 3 表示的几何体/地理数据。
- **ST\_AsGeoJSON** - 新增: 2.0.0 新增 ST\_AsGeoJSON(geometry) 函数。返回几何体或特征体的 GeoJSON 格式。
- **ST\_AsSVG** - 新增: 2.0.0 新增 ST\_AsSVG(geometry) 函数。返回 SVG 路径数据。
- **ST\_EndPoint** - 新增: 2.0.0 新增 ST\_EndPoint(geometry) 函数。返回几何体的终点。PostGIS 2.0.0 之前，对于非点几何体，该函数返回 NULL。2.0.0 版本中，对于非点几何体，该函数返回 NULL。
- **ST\_GDALDrivers** - 新增: 2.0.6, 2.1.3 新增 - GUC 参数 gdal\_enabled\_drivers 控制。返回由 PostGIS 通过 GDAL 支持的栅格格式列表。只有那些 can\_write=True 的格式才能被 ST\_AsGDALRaster 使用。
- **ST\_GeomFromText** - 新增: PostGIS 2.0.0 新增 ST\_GeomFromText('GEOMETRYCOLLECTION EMPTY') 函数。返回空几何体。WKT 格式。PostGIS 2.0.0 之前，SQL/MM 标准要求返回 NULL。
- **ST\_GeometryN** - 新增: 2.0.0 新增 ST\_GeometryN(geometry, n) 函数。返回几何体的第 n 个部分。
- **ST\_IsEmpty** - 新增: PostGIS 2.0.0 新增 ST\_IsEmpty(geometry) 函数。测试几何体是否为空。
- **ST\_Length** - 新增: 2.0.0 新增 ST\_Length(geometry) 函数。返回几何体的长度。对于非线几何体，返回 NULL。
- **ST\_LocateAlong** - 新增: 2.0.0 新增 ST\_LocateAlong(geometry, measure) 函数。返回在几何体上匹配测量值的点(s)。



- **ST\_LocateBetween** - 新增: 2.0.0 新增 ST\_Locate Along\_Measure 函数。返回与指定测量范围匹配的几何部分。
- **ST\_ModEdgeSplit** - 新增: 2.0 新增 ST\_ModEdgesSplit 函数。返回修改后的几何。
- **ST\_NumGeometries** - 新增: 2.0.0 新增 ST\_NumGeometries 函数。返回几何集合中的几何数量。2.0.0 新增 ST\_NumGeometries 函数。返回几何集合中的几何数量。
- **ST\_NumInteriorRings** - 新增: 2.0.0 新增 ST\_NumInteriorRings 函数。返回几何中的内部环数量。
- **ST\_PointN** - 新增: 2.0.0 新增 ST\_PointN 函数。PostGIS 2.0.0 新增 ST\_PointN 函数。返回几何中的第 N 个点。
- **ST\_ScaleX** - 新增: 2.0.0 新增 ST\_ScaleX 函数。返回缩放后的几何。
- **ST\_ScaleY** - 新增: 2.0.0 新增 ST\_ScaleY 函数。返回缩放后的几何。
- **ST\_SetScale** - 新增: 2.0.0 新增 ST\_SetPixelSize 函数。2.0.0 新增 ST\_SetPixelSize 函数。返回缩放后的几何。
- **ST\_StartPoint** - 新增: 2.0.0 新增 ST\_StartPoint 函数。PostGIS 2.0.0 新增 ST\_StartPoint 函数。返回几何中的第一个点。

### 13.12.14 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

- **&&** - 1.5.0 新增 ST\_Equals 函数。A 与 B 是否相等。TRUE 表示相等。
- **PostGIS\_LibXML\_Version** - 1.5 新增 ST\_LibXML\_Version 函数。返回 libxml2 库的版本号。
- **ST\_AddMeasure** - 1.5.0 新增 ST\_AddMeasure 函数。沿线性几何插值测量。
- **ST\_AsBinary** - 1.5.0 新增 ST\_AsBinary 函数。返回 OGC/ISO Well-Known Binary (WKB) 表示的几何/地理数据，不含 SRID 元数据。
- **ST\_AsGML** - 1.5.0 新增 ST\_AsGML 函数。返回 GML 2 或 GML 3 表示的几何。
- **ST\_AsGeoJSON** - 1.5.0 新增 ST\_AsGeoJSON 函数。返回几何或特征在 GeoJSON 格式。
- **ST\_AsText** - 1.5.0 新增 ST\_AsText 函数。返回 WKT(Well-Known Text) 表示的几何，不含 SRID 元数据。
- **ST\_Buffer** - Availability: 1.5 - ST\_Buffer 被增强以支持不同的端盖和连接类型。这些对于将道路线字符串转换为具有平坦或方形边缘的多边形道路非常有用，而不是圆角边缘。Thin wrapper for geography was added. Computes a geometry covering all points within a given distance from a geometry.



- ST\_ClosestPoint** - 1.5.0 `geometry geometry geometry`. 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** - 1.5.0 `geometry collection type`. Given a geometry collection, returns a multi-geometry containing only elements of a specified type.
- ST\_Covers** - 1.5.0 `geometry geometry`. Tests if every point of B lies in A
- ST\_DFullyWithin** - 1.5.0 `geometry geometry float`. Tests if a geometry is entirely inside a distance of another
- ST\_DWithin** - Availability: 1.5.0 support for geography was introduced Tests if two geometries are within a given distance
- ST\_Distance** - 1.5.0 `geometry geometry`. `float` (longest) `float`.
- ST\_DistanceSphere** - 1.5 `geometry geometry`. 1.5 `float` `float`. PostGIS 1.5 `float`.
- ST\_DistanceSpheroid** - 1.5 `geometry geometry`. 1.5 `float` `float`. PostGIS 1.5 `float`.
- ST\_DumpPoints** - 1.5.0 `geometry`. `geometry[]`.
- ST\_Envelope** - 1.5.0 `geometry`, `float4` `float8` `float8`. `double precision; float8` `float8`.
- 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** - 1.5 `xml`. LibXML2 1.6 `xml`. GML `geometry` ST\_Geometry `geometry`. `ST_GeomFromGML` `geometry`.
- ST\_GeomFromGML** - 1.5 `xml`. LibXML2 1.6 `xml`. `GML` `PostGIS` `geometry`.
- ST\_GeomFromKML** - Availability: 1.5, requires libxml2 2.6+ `KML` `PostGIS` `geometry`.
- ST\_HausdorffDistance** - 1.5.0 `geometry geometry`. `float` 3 `float` (shortest) `float`.
- 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** - 1.5.0 `geometry`. `float`.
- ST\_LongestLine** - 1.5.0 `geometry geometry`. `float` 3 `float` (longest) `float`.
- ST\_MakeEnvelope** - 1.5 `float` `float` `SRID` `SRS` `geometry`.
- ST\_MaxDistance** - 1.5.0 `geometry geometry`. `float` 2 `float`.
- ST\_ShortestLine** - 1.5.0 `geometry geometry`. `float` 2 `float`.
- ~=** - 1.5.0 `geometry geometry`. A `B` `TRUE` `FALSE`.

### 13.12.15 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** - 1.4.0 `Populate_Geometry_Columns()`. Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- **ST\_Collect** - 1.4.0 `ST_Collect()`. `ST_MakeLine` `ST_MakeLine()`. `ST_MakeLine` `ST_MakeLine()`. Creates a GeometryCollection or Multi\* geometry from a set of geometries.
- **ST\_ContainsProperly** - 1.4.0 `ST_ContainsProperly()`. Tests if every point of B lies in the interior of A
- **ST\_GeoHash** - 1.4.0 `ST_GeoHash()`. `GeoHash` `GeoHash()`.
- **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** - 1.4.0 `ST_LocateBetweenElevations()`. Returns the portions of a geometry that lie in an elevation (Z) range.
- **ST\_MakeLine** - 1.4.0 `ST_MakeLine()`. `ST_MakeLine` `ST_MakeLine()`. `ST_MakeLine` `ST_MakeLine()`. `ST_MakeLine` `ST_MakeLine()`. `ST_MakeLine` `ST_MakeLine()`.
- **ST\_MinimumBoundingCircle** - 1.4.0 `ST_MinimumBoundingCircle()`. 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.

### 13.12.16 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** - 1.3.2 `geometry geometry AsGML(text encoding_name = 'UTF8')`. Returns GML 2 or GML 3 `text`.
- **ST\_AsGeoJSON** - 1.3.4 `geometry geometry AsGeoJSON(int options = 0)`. Return a geometry or feature in GeoJSON format.
- **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** - 1.3.3 `geometry geometry SimplifyPreserveTopology(double threshold)`. Returns a simplified and valid representation of a geometry, using the Douglas-Peucker algorithm.

# Chapter 14

## Reporting Problems

### 14.1 Reporting Software Bugs

Reporting bugs effectively is a fundamental way to help PostGIS development. The most effective bug report is that enabling PostGIS developers to reproduce it, so it would ideally contain a script triggering it and every information regarding the environment in which it was detected. Good enough info can be extracted running `SELECT postgis_full_version()` [for PostGIS] and `SELECT version()` [for postgresql].

If you aren't using the latest release, it's worth taking a look at its [release changelog](#) first, to find out if your bug has already been fixed.

Using the [PostGIS bug tracker](#) will ensure your reports are not discarded, and will keep you informed on its handling process. Before reporting a new bug please query the database to see if it is a known one, and if it is please add any new information you have about it.

You might want to read Simon Tatham's paper about [How to Report Bugs Effectively](#) before filing a new report.

### 14.2 Reporting Documentation Issues

The documentation should accurately reflect the features and behavior of the software. If it doesn't, it could be because of a software bug or because the documentation is in error or deficient.

Documentation issues can also be reported to the [PostGIS bug tracker](#).

If your revision is trivial, just describe it in a new bug tracker issue, being specific about its location in the documentation.

If your changes are more extensive, a patch is definitely preferred. This is a four step process on Unix (assuming you already have [git](#) installed):

1. Clone the PostGIS' git repository. On Unix, type:

```
git clone https://git.osgeo.org/gitea/postgis/postgis.git
```

This will be stored in the directory `postgis`

2. Make your changes to the documentation with your favorite text editor. On Unix, type (for example):

```
vim doc/postgis.xml
```

Note that the documentation is written in DocBook XML rather than HTML, so if you are not familiar with it please follow the example of the rest of the documentation.

---

3. Make a patch file containing the differences from the master copy of the documentation. On Unix, type:  
**git diff doc/postgis.xml > doc.patch**
  4. Attach the patch to a new issue in bug tracker.
-

# Appendix A

## Appendix

### A.1 PostGIS 3.6.0

2025/09/01

This version requires PostgreSQL 12-18beta3, GEOS 3.8 or higher, and Proj 6.1+. To take advantage of all features, GEOS 3.14+ is needed. To take advantage of all SFCGAL features, SFCGAL 2.2+ is needed.

Many thanks to our translation teams, in particular:

Teramoto Ikuhiro (Japanese Team)

Daniel Nylander (Swedish Team)

Dapeng Wang, Zuo Chenwei from HighGo (Chinese Team)

Denys Kovshun (Ukrainian Team)

#### A.1.1 Breaking Changes

[#5799](#), make ST\_TileEnvelope clips envelopes to tile plane extent (Paul Ramsey)

[#5829](#), remove constraint checking from geometry\_columns view (Paul Ramsey)

[#3373](#), [GT-255](#), [topology] Support for upgrading domains (Ayo Adesugba, U.S. Census Bureau)

[GT-252](#), ST\_NumGeometries/ST\_GeometryN treat TIN and PolyhedralSurface as unitary geometries, use ST\_NumPatches/ST\_PatchN for patch access (Loïc Bartoletti)

[#3110](#), [GT-242](#), [topology] Support for bigint (Ayo Adesugba, U.S. Census Bureau)

[#5359](#), [#5897](#), [GT-260](#) [tiger\_geocoder] Use @extschema:extension@ for PG >= 16 to schema qualify dependent extensions, switch to use typmod for tiger tables (Regina Obe)

#### A.1.2 Removed / Deprecate signatures

[#3110](#), [GT-242](#), [topology] Support for bigint (Ayo Adesugba, U.S. Census Bureau)

[#5498](#) Drop st\_approxquantile(raster, double precision), wasn't usable as it triggered is not unique error when used (Regina Obe)

---

### A.1.3 New Features

**GH-803**, [sfcgal] ADD CG\_Simplify function (Loïc Bartoletti)

**GH-805**, [sfcgal] Add M support for SFCGAL >= 1.5.0 (Loïc Bartoletti)

**GH-801**, [sfcgal] ADD CG\_3DAlphaWrapping function (Jean Felder)

**#5894**, [topology] TotalTopologySize (Sandro Santilli)

**#5890**, [topology] ValidateTopologyPrecision, MakeTopologyPrecise (Sandro Santilli)

**#5861**, [topology] Add --drop-topology switch to pgtopo\_import (Sandro Santilli)

**#1247**, [raster] ST\_AsRasterAgg (Sandro Santilli)

**#5784**, **GT-223** Export circ\_tree\_distance\_tree\_internal for mobilitydb use (Maxime Schoemans)

**GT-228** [sfcgal] Add new functions (Scale, Translate, Rotate, Buffer 3D and Straight Skeleton Partition) from SFCGAL 2 (Loïc Bartoletti)

[raster] New GUC postgis.gdal\_cpl\_debug, enables GDAL debugging messages and routes them into the PostgreSQL logging system. (Paul Ramsey)

**#5841**, Change interrupt handling to remove use of pqsignal to support PG 18 (Paul Ramsey)

Add ST\_CoverageClean to edge match and gap remove polygonal coverages (Paul Ramsey) from GEOS 3.14 (Martin Davis)

**#3110**, **GT-242** [topology] Support for bigint (Ayo Adesugba, U.S. Census Bureau)

[raster] Add ST\_ReclassExact to quickly remap values in raster (Paul Ramsey)

**#5971**, [tiger] Option to build --without-tiger (Regina Obe)

---