

PostGIS 3.6.0alpha1 ☒☒☒☒☒☒

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 (几何) 数据类型	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 与 WKT	33
4.2	Geometry Data Type	34
4.2.1	OpenGIS WKB 与 WKT	34
4.3	PostGIS 数据类型	36
4.3.1	数据类型	37
4.3.2	PostGIS 数据类型	38
4.3.3	数据类型与数据类型	39
4.3.4	数据类型 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 表	45
4.5.1	SPATIAL_REF_SYS Table	45
4.5.2	SPATIAL_REF_SYS 表	46
4.6	几何元数据	47
4.6.1	几何元数据	47
4.6.2	The GEOMETRY_COLUMNS VIEW	48
4.6.3	geometry_columns 表	48
4.7	GIS (GIS) 数据	50
4.7.1	SQL 数据	51
4.7.2	shp2pgsql: ESRI shapefile 数据	51
4.8	GIS 数据	53
4.8.1	SQL 数据	53
4.8.2	数据	54
4.9	GIS 数据	54
4.9.1	GiST 数据	55
4.9.2	GiST 数据	55
4.9.3	GiST 数据	57
4.9.4	数据	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	GIS 数据	68
6.1	GIS 数据	68
6.1.1	数据	68
6.1.2	数据	68
6.2	GIS 数据	69
6.3	数据	69

7	PostGIS Reference	70
7.1	PostgreSQL PostGIS Geometry/Geography/Box 2D	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	管理几何列和表	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	几何构造器 (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	几何访问器 (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	ST_ (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	⊞ (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	运算符 (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	ST_	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
7.14.26T_SetEffectiveArea	430
7.14.28T_TriangulatePolygon	432
7.14.28T_VoronoiLines	434
7.14.29T_VoronoiPolygons	435
7.15Coverages	437
7.15.1ST_CoverageInvalidEdges	437
7.15.2ST_CoverageSimplify	438
7.15.3ST_CoverageUnion	440
7.16Affine Transformations	441
7.16.1ST_Affine	441
7.16.2ST_Rotate	443
7.16.3ST_RotateX	444
7.16.4ST_RotateY	445
7.16.5ST_RotateZ	446
7.16.6ST_Scale	447
7.16.7ST_Translate	448
7.16.8ST_TransScale	449
7.17Clustering Functions	451
7.17.1ST_ClusterDBSCAN	451
7.17.2ST_ClusterIntersecting	453
7.17.3ST_ClusterIntersectingWin	453
7.17.4ST_ClusterKMeans	454
7.17.5ST_ClusterWithin	456
7.17.6ST_ClusterWithinWin	457
7.18Bounding Box Functions	458
7.18.1Box2D	458
7.18.2Box3D	459
7.18.3ST_EstimatedExtent	460
7.18.4ST_Expand	461
7.18.5ST_Extent	462
7.18.6ST_3DExtent	463
7.18.7ST_MakeBox2D	465
7.18.8ST_3DMakeBox	465
7.18.9ST_XMax	466
7.18.10T_XMin	467
7.18.11T_YMax	468
7.18.12T_YMin	469
7.18.13T_ZMax	470

7.18.1	ST_ZMin	471
7.19	￼￼￼ (Linear Referencing)	472
7.19.1	ST_LineInterpolatePoint	472
7.19.2	ST_3DLineInterpolatePoint	473
7.19.3	ST_LineInterpolatePoints	474
7.19.4	ST_LineLocatePoint	475
7.19.5	ST_LineSubstring	476
7.19.6	ST_LocateAlong	478
7.19.7	ST_LocateBetween	479
7.19.8	ST_LocateBetweenElevations	481
7.19.9	ST_InterpolatePoint	482
7.19.10	ST_AddMeasure	482
7.20	Trajectory Functions	483
7.20.1	ST_IsValidTrajectory	483
7.20.2	ST_ClosestPointOfApproach	484
7.20.3	ST_DistanceCPA	485
7.20.4	ST_CPAWithin	486
7.21	Version Functions	487
7.21.1	PostGIS_Extensions_Upgrade	487
7.21.2	PostGIS_Full_Version	488
7.21.3	PostGIS_GEOS_Version	488
7.21.4	PostGIS_GEOS_Compiled_Version	489
7.21.5	PostGIS_Liblwgeom_Version	489
7.21.6	PostGIS_LibXML_Version	490
7.21.7	PostGIS_LibJSON_Version	490
7.21.8	PostGIS_Lib_Build_Date	491
7.21.9	PostGIS_Lib_Version	491
7.21.10	PostGIS_PROJ_Version	492
7.21.11	PostGIS_PROJ_Compiled_Version	492
7.21.12	PostGIS_Wagyu_Version	493
7.21.13	PostGIS_Scripts_Build_Date	494
7.21.14	PostGIS_Scripts_Installed	494
7.21.15	PostGIS_Scripts_Released	495
7.21.16	PostGIS_Version	495
7.22	PostGIS GUC(Grand Unified Custom Variable)	496
7.22.1	postgis.gdal_datapath	496
7.22.2	postgis.gdal_enabled_drivers	497
7.22.3	postgis.enable_outdb_rasters	498
7.22.4	postgis.gdal_vsi_options	499

7.22.5	postgis.gdal_cpl_debug	500
7.23	Troubleshooting Functions	500
7.23.1	PostGIS_AddBBox	500
7.23.2	PostGIS_DropBBox	501
7.23.3	PostGIS_HasBBox	501
8	SFCGAL Functions Reference	503
8.1	SFCGAL Management Functions	503
8.1.1	postgis_sfcgal_version	503
8.1.2	postgis_sfcgal_full_version	503
8.2	SFCGAL Accessors and Setters	504
8.2.1	CG_ForceLHR	504
8.2.2	CG_IsPlanar	504
8.2.3	CG_IsSolid	505
8.2.4	CG_MakeSolid	505
8.2.5	CG_Orientation	506
8.2.6	CG_Area	506
8.2.7	CG_3DArea	507
8.2.8	CG_Volume	507
8.2.9	ST_ForceLHR	508
8.2.10	ST_IsPlanar	509
8.2.11	ST_IsSolid	509
8.2.12	ST_MakeSolid	510
8.2.13	ST_Orientation	510
8.2.14	ST_3DArea	511
8.2.15	ST_Volume	512
8.3	SFCGAL Processing and Relationship Functions	513
8.3.1	CG_Intersection	513
8.3.2	CG_Intersects	514
8.3.3	CG_3DIntersects	514
8.3.4	CG_Difference	515
8.3.5	ST_3DDifference	516
8.3.6	CG_3DDifference	517
8.3.7	CG_Distance	518
8.3.8	CG_3DDistance	519
8.3.9	ST_3DConvexHull	520
8.3.10	CG_3DConvexHull	520
8.3.11	ST_3DIntersection	521
8.3.12	CG_3DIntersection	522

8.3.13CG_Union	524
8.3.14ST_3DUnion	525
8.3.15CG_3DUnion	525
8.3.16ST_AlphaShape	527
8.3.17CG_AlphaShape	527
8.3.18CG_ApproxConvexPartition	530
8.3.19ST_ApproximateMedialAxis	531
8.3.20CG_ApproximateMedialAxis	532
8.3.21ST_ConstrainedDelaunayTriangles	533
8.3.22CG_ConstrainedDelaunayTriangles	534
8.3.23ST_Extrude	535
8.3.24CG_Extrude	535
8.3.25CG_ExtrudeStraightSkeleton	537
8.3.26CG_GreeneApproxConvexPartition	538
8.3.27ST_MinkowskiSum	539
8.3.28CG_MinkowskiSum	540
8.3.29ST_OptimalAlphaShape	542
8.3.30CG_OptimalAlphaShape	543
8.3.31CG_OptimalConvexPartition	545
8.3.32CG_StraightSkeleton	546
8.3.33ST_StraightSkeleton	548
8.3.34ST_Tesselate	549
8.3.35CG_Tesselate	550
8.3.36CG_Triangulate	552
8.3.37CG_Visibility	553
8.3.38CG_YMonotonePartition	554
8.3.39CG_StraightSkeletonPartition	555
8.3.40CG_3DBuffer	556
8.3.41CG_Rotate	558
8.3.42CG_2DRotate	559
8.3.43CG_3DRotate	559
8.3.44CG_RotateX	560
8.3.45CG_RotateY	561
8.3.46CG_RotateZ	561
8.3.47CG_Scale	562
8.3.48CG_3DScale	562
8.3.49CG_3DScaleAroundCenter	563
8.3.50CG_Translate	563
8.3.51CG_3DTranslate	564
8.3.52CG_Simplify	564
8.3.53CG_3DAlphaWrapping	567

9 拓扑 (topology)	571
9.1 拓扑	571
9.1.1 getfaceedges_returntype	571
9.1.2 TopoGeometry	572
9.1.3 validate_topology_returntype	572
9.2 拓扑函数	573
9.2.1 TopoElement	573
9.2.2 TopoElementArray	573
9.3 拓扑 TopoGeometry 函数	574
9.3.1 AddTopoGeometryColumn	574
9.3.2 RenameTopoGeometryColumn	575
9.3.3 DropTopology	576
9.3.4 RenameTopology	576
9.3.5 DropTopoGeometryColumn	577
9.3.6 Populate_Topology_Layer	577
9.3.7 TopologySummary	578
9.3.8 ValidateTopology	579
9.3.9 ValidateTopologyRelation	582
9.3.10 ValidateTopologyPrecision	582
9.3.11 MakeTopologyPrecise	583
9.3.12 FindTopology	583
9.3.13 FindLayer	584
9.3.14 TotalTopologySize	585
9.4 Topology Statistics Management	585
9.5 拓扑函数	586
9.5.1 CreateTopology	586
9.5.2 CopyTopology	587
9.5.3 ST_InitTopoGeo	587
9.5.4 ST_CreateTopoGeo	588
9.5.5 TopoGeo_AddPoint	589
9.5.6 TopoGeo_AddLineString	589
9.5.7 TopoGeo_AddPolygon	590
9.5.8 TopoGeo_LoadGeometry	590
9.6 拓扑函数	591
9.6.1 ST_AddIsoNode	591
9.6.2 ST_AddIsoEdge	592
9.6.3 ST_AddEdgeNewFaces	592
9.6.4 ST_AddEdgeModFace	593
9.6.5 ST_RemEdgeNewFace	594

9.6.6	ST_RemEdgeModFace	594
9.6.7	ST_ChangeEdgeGeom	595
9.6.8	ST_ModEdgeSplit	596
9.6.9	ST_ModEdgeHeal	597
9.6.10	ST_NewEdgeHeal	597
9.6.11	ST_MoveIsoNode	598
9.6.12	ST_NewEdgesSplit	598
9.6.13	ST_RemoveIsoNode	599
9.6.14	ST_RemoveIsoEdge	600
9.7	拓扑几何	600
9.7.1	GetEdgeByPoint	600
9.7.2	GetFaceByPoint	601
9.7.3	GetFaceContainingPoint	602
9.7.4	GetNodeByPoint	603
9.7.5	GetTopologyID	604
9.7.6	GetTopologySRID	604
9.7.7	GetTopologyName	605
9.7.8	ST_GetFaceEdges	605
9.7.9	ST_GetFaceGeometry	606
9.7.10	GetRingEdges	607
9.7.11	GetNodeEdges	607
9.8	拓扑操作	608
9.8.1	Polygonize	608
9.8.2	AddNode	609
9.8.3	AddEdge	609
9.8.4	AddFace	611
9.8.5	ST_Simplify	612
9.8.6	RemoveUnusedPrimitives	613
9.9	TopoGeometry 类型	613
9.9.1	CreateTopoGeom	613
9.9.2	toTopoGeom	615
9.9.3	TopoElementArray_Agg	616
9.9.4	TopoElement	617
9.10	TopoGeometry 函数	617
9.10.1	clearTopoGeom	617
9.10.2	TopoGeom_addElement	618
9.10.3	TopoGeom_remElement	618
9.10.4	TopoGeom_addTopoGeom	619
9.10.5	toTopoGeom	620

9.11 TopoGeometry 函数	620
9.11.1 GetTopoGeomElementArray	620
9.11.2 GetTopoGeomElements	620
9.11.3 ST_SRID	621
9.12 TopoGeometry 函数	622
9.12.1 AsGML	622
9.12.2 AsTopoJSON	624
9.13 空间关系函数	626
9.13.1 Equals	626
9.13.2 Intersects	626
9.14 Importing and exporting Topologies	627
9.14.1 Using the Topology exporter	627
9.14.2 Using the Topology importer	628
10 空间数据库管理, 数据库	629
10.1 空间数据库管理	629
10.1.1 raster2pgsql 函数	629
10.1.1.1 Example Usage	629
10.1.1.2 raster2pgsql options	630
10.1.2 PostGIS 空间数据库管理	631
10.1.3 Using "out db" cloud rasters	632
10.2 空间数据库管理	633
10.2.1 空间数据库管理	633
10.2.2 空间数据库管理	634
10.3 PostGIS 空间数据库管理	635
10.3.1 空间数据库管理 ST_AsPNG 函数 PHP 函数	635
10.3.2 空间数据库管理 ST_AsPNG 函数 ASP.NET C# 函数	636
10.3.3 空间数据库管理 函数 Java 函数	637
10.3.4 PLPython 函数 SQL 函数	639
10.3.5 PSQL 函数	639
11 空间数据库管理	641
11.1 空间数据库管理	642
11.1.1 geomval	642
11.1.2 addbandarg	642
11.1.3 rastbandarg	642
11.1.4 raster	643
11.1.5 reclassarg	643
11.1.6 summarystats	644

11.1.7unionarg	644
11.2 数据类型	645
11.2.1AddRasterConstraints	645
11.2.2DropRasterConstraints	647
11.2.3AddOverviewConstraints	648
11.2.4DropOverviewConstraints	649
11.2.5PostGIS_GDAL_Version	649
11.2.6PostGIS_Raster_Lib_Build_Date	650
11.2.7PostGIS_Raster_Lib_Version	650
11.2.8ST_GDALDrivers	651
11.2.9ST_Contour	656
11.2.10ST_InterpolateRaster	657
11.2.11UpdateRasterSRID	657
11.2.12ST_CreateOverview	658
11.3 数据类型 (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 数据类型 (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

11.4.1	ST_SkewX	683
11.4.1	ST_SkewY	684
11.4.1	ST_SRID	685
11.4.1	ST_Summary	685
11.4.1	ST_UpperLeftX	686
11.4.2	ST_UpperLeftY	687
11.4.2	ST_Width	687
11.4.2	ST_WorldToRasterCoord	688
11.4.2	ST_WorldToRasterCoordX	689
11.4.2	ST_WorldToRasterCoordY	689
11.5	ST_MetaData	690
11.5.1	ST_BandMetaData	690
11.5.2	ST_BandNoDataValue	692
11.5.3	ST_BandIsNoData	692
11.5.4	ST_BandPath	694
11.5.5	ST_BandFileSize	694
11.5.6	ST_BandFileTimestamp	695
11.5.7	ST_BandPixelType	695
11.5.8	ST_MinPossibleValue	696
11.5.9	ST_HasNoBand	697
11.6	ST_PixelAsPolygon (setter)	697
11.6.1	ST_PixelAsPolygon	697
11.6.2	ST_PixelAsPolygons	698
11.6.3	ST_PixelAsPoint	699
11.6.4	ST_PixelAsPoints	700
11.6.5	ST_PixelAsCentroid	701
11.6.6	ST_PixelAsCentroids	701
11.6.7	ST_Value	703
11.6.8	ST_NearestValue	706
11.6.9	ST_SetZ	707
11.6.1	ST_SetM	709
11.6.1	ST_Neighborhood	710
11.6.1	ST_SetValue	712
11.6.1	ST_SetValues	713
11.6.1	ST_DumpValues	721
11.6.1	ST_PixelOfValue	722
11.7	ST_SetGeoReference	724
11.7.2	ST_SetRotation	725

11.7.3	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	ST_Band	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	ST_Stats	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	ST_As*	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	ST_MapAlgebra	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

11.12.6T_MapAlgebra (expression version)	779
11.12.3T_MapAlgebraExpr	782
11.12.8T_MapAlgebraExpr	784
11.12.9T_MapAlgebraFct	789
11.12.5T_MapAlgebraFct	793
11.12.1T_MapAlgebraFctNgb	797
11.12.1T_Reclass	799
11.12.1T_Union	801
11.13.1T_Distinct4ma	802
11.13.1T_InvDistWeight4ma	803
11.13.1T_Max4ma	804
11.13.1T_Mean4ma	805
11.13.1T_Min4ma	807
11.13.6T_MinDist4ma	808
11.13.1T_Range4ma	809
11.13.8T_StdDev4ma	810
11.13.9T_Sum4ma	811
11.14.1T_Aspect	812
11.14.1T_HillShade	813
11.14.1T_Roughness	815
11.14.1T_Slope	816
11.14.1T_TPI	818
11.14.6T_TRI	818
11.15.1T_Box3D	819
11.15.1T_ConvexHull	820
11.15.1T_DumpAsPolygons	821
11.15.1T_Envelope	822
11.15.1T_MinConvexHull	822
11.15.6T_Polygon	824
11.16.1T_&&	825
11.16.1T_&<	826
11.16.1T_&>	826
11.16.4	827
11.16.1T_@	828
11.16.6=	828

11.16.7	829
11.17 空間關係函數	829
11.17.\$T_Contains	829
11.17.\$T_ContainsProperly	830
11.17.\$T_Covers	831
11.17.\$T_CoveredBy	832
11.17.\$T_Disjoint	833
11.17.\$T_Intersects	834
11.17.\$T_Overlaps	835
11.17.\$T_Touches	836
11.17.\$T_SameAlignment	837
11.17.\$T_NotSameAlignmentReason	838
11.17.\$T_Within	839
11.17.\$T_DWithin	840
11.17.\$T_DFullyWithin	841
11.18 快速提示	842
11.18.0 Out-DB Rasters	842
11.18.1.0 Directory containing many files	842
11.18.1.1 Maximum Number of Open Files	842
11.18.1.2.1 Maximum number of open files for the entire system	843
11.18.1.2.2 Maximum number of open files per process	843
12 PostGIS Extras	845
12.1 空間索引	845
12.1.1 空間索引	845
12.1.2 空間索引	846
12.1.2.1 stdaddr	846
12.1.3 空間索引	846
12.1.3.1 rules table	846
12.1.3.2 lex table	849
12.1.3.3 gaz table	850
12.1.4 空間索引	850
12.1.4.1 debug_standardize_address	850
12.1.4.2 parse_address	852
12.1.4.3 standardize_address	853
12.2 TIGER 空間索引	854
12.2.1 Drop_Indexes_Generate_Script	855
12.2.2 Drop_Nation_Tables_Generate_Script	856
12.2.3 Drop_State_Tables_Generate_Script	856

12.2.4	Geocode	857
12.2.5	Geocode_Intersection	860
12.2.6	Get_Geocode_Setting	861
12.2.7	Get_Tract	862
12.2.8	Install_Missing_Indexes	863
12.2.9	Loader_Generate_Census_Script	863
12.2.10	Loader_Generate_Script	865
12.2.11	Loader_Generate_Nation_Script	867
12.2.12	Missing_Indexes_Generate_Script	868
12.2.13	Normalize_Address	869
12.2.14	Page_Normalize_Address	870
12.2.15	Print_Addy	872
12.2.16	Reverse_Geocode	873
12.2.17	Topology_Load_Tiger	875
12.2.18	Set_Geocode_Setting	877

13 PostGIS Special Functions Index 879

13.1	PostGIS Aggregate Functions	879
13.2	PostGIS Window Functions	880
13.3	PostGIS SQL-MM Compliant Functions	880
13.4	PostGIS Geography Support Functions	884
13.5	PostGIS Raster Support Functions	886
13.6	PostGIS Geometry / Geography / Raster Dump Functions	891
13.7	PostGIS Box Functions	892
13.8	PostGIS Functions that support 3D	893
13.9	PostGIS Curved Geometry Support Functions	899
13.10	PostGIS Polyhedral Surface Support Functions	902
13.11	PostGIS Function Support Matrix	906
13.12	New, Enhanced or changed PostGIS Functions	924
13.12.1	PostGIS Functions new or enhanced in 3.6	924
13.12.2	PostGIS Functions new or enhanced in 3.5	925
13.12.3	PostGIS Functions new or enhanced in 3.4	927
13.12.4	PostGIS Functions new or enhanced in 3.3	929
13.12.5	PostGIS Functions new or enhanced in 3.2	930
13.12.6	PostGIS Functions new or enhanced in 3.1	931
13.12.7	PostGIS Functions new or enhanced in 3.0	933
13.12.8	PostGIS Functions new or enhanced in 2.5	934
13.12.9	PostGIS Functions new or enhanced in 2.4	936
13.12.10	PostGIS Functions new or enhanced in 2.3	937

13.12.1	PostGIS Functions new or enhanced in 2.2	939
13.12.2	PostGIS Functions new or enhanced in 2.1	942
13.12.3	PostGIS Functions new or enhanced in 2.0	948
13.12.4	PostGIS Functions new or enhanced in 1.5	960
13.12.5	PostGIS Functions new or enhanced in 1.4	961
13.12.6	PostGIS Functions new or enhanced in 1.3	962
14	Reporting Problems	963
14.1	Reporting Software Bugs	963
14.2	Reporting Documentation Issues	963
A	Appendix	965
A.1	PostGIS 3.6.0alpha1	965
A.1.1	Breaking Changes	965
A.1.2	New Features	965

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 3.6.0alpha1 是 PostgreSQL 数据库的 GIS 扩展。



PostGIS 是 PostgreSQL 数据库的 GIS 扩展。PostGIS 使用 GiST 和 R-Tree 索引, GIS 数据存储在 PostgreSQL 数据库中。
<https://postgis.net>

Chapter 1

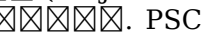


PostGIS is a spatial extension for the PostgreSQL relational database that was created by Refractions Research Inc, as a spatial database technology research project. Refractions is a GIS and database consulting company in Victoria, British Columbia, Canada, specializing in data integration and custom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

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

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

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

PostGIS is a community project, and we would like to thank the following organizations for their support.

- [Aiven](#)
- [Arrival 3D](#)
- [Associazione Italiana per l'Informazione Geografica Libera \(GFOSS.it\)](#)
- [AusVet](#)
- [Avencia](#)
- [Azavea](#)
- [Boundless](#)
- [Cadcorp](#)

- PostGIS 2.0.0
PledgBank

postgistopology - 10 依赖库 2.0.0 依赖库 toTopGeometry 依赖库
250 依赖库。

postgis64windows - 20 依赖库 64 依赖库 PostGIS 依赖库 100
依赖库, 依赖库。 依赖库 PostgreSQL 依赖库 PostGIS 2.0.1 64 依赖库
依赖库。

依赖库 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

依赖库, PostGIS 依赖库 **PostgreSQL DBMS** - PostGIS 依赖库
依赖库 PostgreSQL 依赖库, 依赖库, GiST 依赖库, 依赖库 SQL 依赖库
依赖库。

Chapter 2

PostGIS

PostGIS is a spatial database engine for PostgreSQL.

2.1

PostGIS is a spatial database engine for PostgreSQL.

```
tar -xvzf postgis-3.6.0alpha1.tar.gz
cd postgis-3.6.0alpha1
./configure
make
make install
```

PostGIS is a spatial database engine for PostgreSQL. (Section 3.3) (Section 3.4)

2.2

Note

PostGIS is a spatial database engine for PostgreSQL. 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](#).



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](#) [very bleeding-edge windows experimental builds](#) [PostGIS](#).

The PostGIS module is an extension to the PostgreSQL backend server. As such, PostGIS 3.6.0alpha1 *requires* full PostgreSQL server headers access in order to compile. It can be built against PostgreSQL versions 12 - 17. Earlier versions of PostgreSQL are *not* supported.

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

Note

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://download.osgeo.org/postgis/source/postgis-3.6.0alpha1.tar.gz> 编译选项。

```
wget https://download.osgeo.org/postgis/source/postgis-3.6.0alpha1.tar.gz
tar -xvzf postgis-3.6.0alpha1.tar.gz
cd postgis-3.6.0alpha1
```

使用 postgis-3.6.0alpha1 (X) 编译选项。

使用, svn 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 postgis-3.6.0alpha1 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 - 17. A complete installation of PostgreSQL (including server headers) is required. PostgreSQL is available from <https://www.postgresql.org> .
For a full PostgreSQL / PostGIS support matrix and PostGIS/GEOS support matrix refer to <https://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS>
- GNU C 编译器 (gcc). PostGIS 使用 ANSI C 编译器 gcc 编译选项。
- GNU Make(gmake 或 make). 使用 GNU make 或 make 编译选项。make -v 编译选项。编译选项 make 或 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 3.12+ 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) <http://www.docbook.org/> DocBook <http://www.docbook.org/>
- DBLatex(dblatex) <http://dblatex.sourceforge.net/> PDF <http://dblatex.sourceforge.net/>
- ImageMagick(convert) <http://www.imagemagick.org/> ImageMagick <http://www.imagemagick.org/>

2.2.3 Building

The Makefile is located in the `Makefile` directory. The Makefile is used to build the PostGIS binaries.

./configure

The `./configure` script is used to configure the build environment. It checks for the presence of the required libraries and headers, and sets the appropriate compiler and linker flags.

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

```
--prefix=PREFIX PostgreSQL  SQL .  PostgreSQL 
.
```



PostgreSQL 9.1.3 64-bit
<http://trac.osgeo.org/postgis/ticket/635>

```
--with-gettext=no PostgreSQL gettext 1.10.1, 1.10.1
gettext 1.10.1, 1.10.1. 1.10.1
http://trac.osgeo.org/postgis/ticket/748 1.10.1.
: gettext 1.10.1, 1.10.1. gettext 1.10.1, 1.10.1
GUI 1.10.1/1.10.1.
```

--with-sfcgal=PATH 指定 PostGIS 的 sfcgal 库的路径。PATH 是 sfcgal-config 脚本返回的路径。

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

Note

PostGIS 的 SVN 仓库，包含最新的源代码。

**./autogen.sh**

运行 **configure** 脚本，生成 PostGIS 的 Makefile。

运行 **tar** 命令解压 PostGIS 的源代码，运行 **configure** 脚本，运行 **./autogen.sh** 脚本。

2.2.4 编译

运行 Makefile 编译 PostGIS 的源代码。

make

运行 **make** 命令，编译 PostGIS 的源代码。

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 的源代码。包含 PostGIS 的源代码 (cheat sheet) html 文件。运行 **xsltproc** 命令，生成 doc 文件夹中的 topology_cheatsheet.html, tiger_geocoder_cheatsheet.html, raster_cheatsheet.html, postgis_cheatsheet.html 4 个 HTML 文件。

运行 **pdf** 命令，生成 PostGIS / PostgreSQL Study Guides 的 PDF 文件。

make cheatsheets

2.2.5 PostGIS Extensions 安装

PostgreSQL 9.1 的 PostGIS 扩展安装。

运行 **function descriptions** 命令，生成 docbook 文件。

make comments

运行 **tar** 命令解压 PostGIS 的源代码，运行 **tar** 命令解压 comments 文件。

运行 PostgreSQL 9.1 的 extensions 命令，安装 PostGIS 的扩展。

```
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 安装 OS 安装 PostGIS 安装。 PostGIS binaries 安装。

extension 安装 PostgreSQL 安装 PostgreSQL / share / extension 安装 extensions 安装 PostGIS 安装。

- extension 安装 extension 安装. postgres.control, postgres_topology.control.
- extension 安装 /sql 安装. postgres 安装 share/extension 安装 extensions/postgis/sql/*.sql, extensions/postgis_topology/sql/*.sql

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

psql 安装。

```
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	3.6.0alpha1	3.6.0alpha1
address_standardizer_data_us	3.6.0alpha1	3.6.0alpha1
postgis	3.6.0alpha1	3.6.0alpha1
postgis_raster	3.6.0alpha1	3.6.0alpha1
postgis_sfcgal	3.6.0alpha1	
postgis_tiger_geocoder	3.6.0alpha1	3.6.0alpha1
postgis_topology	3.6.0alpha1	

(6 rows)

extension 安装, installed_version 安装。 postgres extension 安装。 PgAdmin III 1.14 安装 extensions 安装。

extension 安装 pgAdmin extension 安装 sql 安装 postgres extension 安装:

```
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 安装完成, 安装完成, 安装完成.

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

List of installed extensions

```
-[ RECORD 1 ]-----
Name          | postgis
Version       | 3.6.0alpha1
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       | 3.6.0alpha1
Schema        | tiger
Description    | PostGIS tiger geocoder and reverse geocoder
-[ RECORD 4 ]-----
Name          | postgis_topology
Version       | 3.6.0alpha1
Schema        | topology
Description    | PostGIS topology spatial types and functions
```

Warning



spatial_ref_sys, layer, topology 安装失败。 安装 postgis 和 postgis_topology 扩展失败。 安装失败。 PosGIS 2.0.1 安装失败。 srid 安装失败。 安装失败。 trac 安装失败。 extension 安装失败。 CREATE EXTENSION 安装失败。 PostgreSQL extension 安装失败。

安装失败 3.6.0alpha1 安装失败, 安装失败。 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 安装, 安装。

make check

PostgreSQL 安装失败。 安装失败。

**Note**

PostgreSQL, GEOS, 以及 Proj4 的库路径, LD_LIBRARY_PATH 环境变量
需要设置。

**Caution**

因此, **make check** 需要 PATH 包含 PGPORT 环境变量。 PostgreSQL 以及
需要 **--with-pgconfig** 选项。 因此, PostgreSQL 以及
需要 PATH 环境变量。

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

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

.
.
.

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

.
.
.

Running tests

.
.
.

Run tests: 134
Failed: 0

-- if you build with SFCGAL

.
.
.

Running tests

.
.
.

Run tests: 13
Failed: 0

-- if you built with raster support

.
```



```
.  
.  
  
Run Summary:
```

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

```
.  
. .  
  
Running tests  
  
. . .  
  
Run tests: 101  
Failed: 0  
  
-- topology regress  
  
. . .  
  
Running tests  
  
. . .  
  
Run tests: 51  
Failed: 0  
  
-- if you built --with-gui, you should see this too  
  
CUnit - A unit testing framework for C - Version 2.1-2  
http://cunit.sourceforge.net/  
  
. . .  
  
Run Summary:
```

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

```
postgis_tiger_geocoder [X] address_standardizer [XXXXXX]  
installcheck) [XXXXXXXX]. XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
[XXXXXXXX] make install [XXXXXXXXXXXXXXXXXXXXXXXXXXXX].  
address_standardizer [XXX]:  
  
cd extensions/address_standardizer  
make install  
make installcheck  
  
XXXXXXXXXXXXXXXXXXXX:  

```

```

===== 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 数据加载器, 使用 PostgreSQL 的 PostGIS 的 fuzzystmatch 和 address_standardizer 扩展。 address_standardizer 是 PostGIS 的 address_standardizer 扩展。

```

cd extensions/postgis_tiger_geocoder
make install
make installcheck

```

安装和测试:

```

===== 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 安装

PostGIS 安装。

make install

使用 **--prefix** 选项指定 PostgreSQL 的安装路径。

- loader 位于 [prefix]/bin 目录下。
- postgis.sql 位于 [prefix]/share/contrib 目录下。
- PostGIS 位于 [prefix]/lib 目录下。

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 used to standardize addresses. PostGIS 2.2 introduced the `address_standardizer` extension. See Section 12.1 for more details.

The `Normalize Address` extension in PostGIS uses the TIGER geocoder (geocoder) to standardize addresses. See Section 2.4.2 for more details. The geocoder is used to standardize building blocks (building block) and addresses.

The `PCRE` library is required. PCRE is a regular expression engine. See <http://www.pcre.org> for more details. Section 2.2.3 describes PCRE. To install PCRE, run `--with-pcre=/path/to/pcre` and `/path/to/pcre` to the PCRE include and lib directories.

PostGIS 2.1 introduced the `address_standardizer` extension. To install the `CREATE EXTENSION` command, run:

```
CREATE EXTENSION address_standardizer;
```

```
CREATE EXTENSION address_standardizer;
```

rules, gaz, lex files are required.

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

The result 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. `gisdata`, `gisdata_loader_variables`, `gisdata_loader_variables` PC `gisdata` `gisdata_loader_variables`. `TIGER` `gisdata_loader_variables`. `gisdata_loader_variables`, `gisdata_loader_variables`, `tiger.loader_variables` `staging_fold` `gisdata_loader_variables`.
7. `gisdata` `staging_fold` `temp` `gisdata_loader_variables`. `TIGER` `temp` `gisdata_loader_variables`.

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)
```

- 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. □□□□□□□□□□□□□□.

```
cd /gisdata
sh ma load.sh
```

15. TIGER (stat) .

```
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 TIGER

The `Normalize_Address` address standardizer
 takes an address and returns a normalized address.
 See Section 2.3 for more details.

```

postgis_tiger_geocoder |> summarise(
  Normalize_Address = sum(
    pagc_normalize_address = 1,
    tiger = 1,
    rules_table = 1,
    gaz_table = 1,
    lex_table = 1
  )
)

```

2.4.3 Required tools for tiger data loading

```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXXXXXXXXXXXXXXX.
XXXXXX tiger XXXXXXXXXXXXXXXXXXXX, Drop_State_Tables_Generate_ScriptXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.

```

[illegible]

- 7-Zip 是 Windows 和 Linux 上最流行的压缩软件之一。
Unix 上使用 `unzip` 命令解压。
Windows 下，7-Zip/WinRAR 7-zip 见 <http://www.7-zip.org/>

- PostGIS 安装 shp2pgsql 工具
- 使用 `wget` 在 Unix/Linux 上安装 PostGIS。

安装 PostGIS 请访问 <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.

安装 PostGIS 后，您需要安装 **Loader_Generate_Script**。该脚本将生成一个 loader 脚本，用于加载国家范围内的数据。该脚本应仅执行一次，用于升级（从先前的年份 tiger 人口普查数据）或新安装。

安装 PostGIS 后，您需要安装 **Install_Missing_Indexes**：

```
SELECT install_missing_indexes();
```

安装 PostGIS 后，您需要安装 **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;
```

安装 nation 表并 drop 旧表。SQL 脚本 drop 旧表。安装 **Drop_Nation_Tables_Generate_Script**。

```
SELECT drop_nation_tables_generate_script();
```

drop SQL 脚本。

安装 **Loader_Generate_Nation_Script**。该脚本将生成一个 loader 脚本，用于加载国家范围内的数据。该脚本应仅执行一次，用于升级（从先前的年份 tiger 人口普查数据）或新安装。

```
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 安装 PostGIS 3.6.0alpha1

安装 PostGIS 3.6.0alpha1 的步骤如下：

1. PostgreSQL 12 `sudo apt-get install postgresql postgresql-contrib`. PostgreSQL `sudo apt-get install postgresql postgresql-contrib`. (Linux) PostgreSQL `sudo apt-get install postgresql postgresql-contrib`, PostgreSQL `sudo apt-get install postgresql postgresql-contrib`. PostGIS `sudo apt-get install postgresql-12-postgis postgresql-12-postgis-scripts`. `psql` `psql`:
`psql`:

```
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-3.4/

# 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 "3.6.0alpha1";  
ALTER EXTENSION postgis_topology UPDATE TO "3.6.0alpha1";
```

If you get an error notice something like:

```
No migration path defined for b'...'b' to 3.6.0alpha1
```

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 "3.6.0alpha1" 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 "3.6.0alpha1next";  
ALTER EXTENSION postgis_topology UPDATE TO "3.6.0alpha1next";
```

**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'...'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-3.6.0alpha1/*_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) Geometry

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 对浮点精度没有要求。精度为 1E-8 或更高。

4.1.2.1 CircularString

CIRCULARSTRING 是 LINESTRING 的子类型。它由一个或多个圆弧组成，每个圆弧由三个点定义。每个点由一对坐标 (X, Y) 表示。圆弧的半径由第三个点与前两个点的距离决定。如果第三个点与前两个点的距离相等，则圆弧是一个半圆。如果第三个点与前两个点的距离不相等，则圆弧是一个圆的一部分。CIRCULARSTRING 的语法如下：

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

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

4.1.2.2 CompoundCurve

COMPOUNDCURVE (compound curve) 是由一个或多个 CIRCULARSTRING 组成的。每个 CIRCULARSTRING 由三个点定义。COMPOUNDCURVE 的语法如下：

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

4.1.2.3 CurvePolygon

CURVEPOLYGON 是由一个或多个 COMPOUNDCURVE 组成的。每个 COMPOUNDCURVE 由三个点定义。CURVEPOLYGON 的语法如下：

PostGIS 1.4 支持 CURVEPOLYGON。

```
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 数据类型, 存储由一个或多个弧段组成的曲线。

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

4.1.2.5 MultiSurface

MULTISURFACE 数据类型, (几何) 存储由一个或多个曲面组成的表面。

```
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 与 WKT

OpenGIS 规范定义了两种几何数据类型: Well-Known Text (WKT) 和 Well-Known Binary (WKB)。WKT 和 WKB 是两种不同的几何数据表示方法。

WKT(Well-Known Text) 是一种文本格式。WKT SRS 是指 WKT 的坐标系。

- 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 4,0 4,0 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 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))
- GEOMETRYCOLLECTION(POINT(2 3),LINESTRING(2 3,3 4))
- 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 规范定义了两种几何数据类型: Well-Known Text (WKT) 和 Well-Known Binary (WKB)。

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

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

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

```
SELECT 'SRID=4;POINT(0 0) '::geometry;
 geometry
-----
0101000020040000000000000000000000000000000000000000000000000000
```

PostGIS EWKT output has a few differences to OGC WKT:

- For 3DZ geometries the Z qualifier is omitted:
POINT(0 0)
POINT(0 0)
- For 3DM geometries the M qualifier is included:
POINT(0 0)
POINT(0 0)
- For 4D geometries the ZM qualifier is omitted:
POINT(0 0)
POINT(0 0)

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

- POINT(0 0)
- POINT(0 0)
- POINT(0 0)



Caution

PostGIS 与 OGC 规范 (WKB/WKT 与 EWKB/EWKT) 存在差异。OGC 与 PostGIS 规范存在差异。请仔细阅读！

PostGIS 规范 (WKT) 与 OGC 规范存在差异:

- POINT(0 0 0) -- XYZ
- SRID=32632;POINT(0 0) -- SRID 与 XY
- POINTM(0 0 0) -- XYM
- POINT(0 0 0 0) -- XYZM
- SRID=4326;MULTIPOINTM(0 0 0,1 2 1) -- SRID 与 XYM

- MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))
- POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
- MULTIPOLYGON((((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)),((-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))
- GEOMETRYCOLLECTIONM(POINTM(2 3 9), LINESTRINGM(2 3 4, 3 4 5))
- MULTICURVE((0 0, 5 5), CIRCULARSTRING(4 0, 4 4, 8 4))
- POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))
- TRIANGLE ((0 0, 0 9, 9 0, 0 0))
- TIN(((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 0 0 0)))

[illegible]

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

PostGIS

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

4.3 PostGIS ☒☒☒☒☒

例 1. 已知 $\triangle ABC$ 中, $\angle A = 60^\circ$, $\angle B = 45^\circ$, $\angle C = 75^\circ$, $AB = 1$, 求 BC 的长.

PostGIS 是 PostgreSQL 数据库的扩展。它提供了对 PostgreSQL 数据库的地理信息数据的支持。PostGIS 是 PostgreSQL 数据库的扩展，它提供了对 PostgreSQL 数据库的地理信息数据的支持。

PostGIS 大圆 (大圈; great circle arc) (点, 线, 面, 体) 球形形状 (spheroidal shape), 球形形状, 球形形状.

XXXXXXXXXXXXXXXXXXXXX, XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX. XXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.

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

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

4.3.1 创建地理表

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

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

The geography type supports two optional type modifiers:

- POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON. POINT Z, M 或 ZM 指定垂直和测量值。LINESTRINGM 指定测量值。3 指定 3D 点。POINTZM 指定垂直和测量值。
- the SRID modifier restricts the spatial reference system SRID to a particular number. If omitted, the SRID defaults to 4326 (WGS84 geodetic), and all calculations are performed using WGS84.

Examples of creating tables with geography columns:

- POINT: 2D 点:

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

- POINT: 2D 点:

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

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

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

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

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

- POINT: 2D 多边形:

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

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

```
SELECT * FROM geography_columns;
```

PostGIS 3.6.0alpha1 简体中文

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


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 性能优化建议

PostGIS 的性能很大程度上依赖于底层的数据库引擎和硬件配置。以下是一些基本的性能优化建议。

1. 数据库引擎选择：PostGIS 支持多种数据库引擎，如 PostgreSQL、MySQL、Oracle 等。不同的引擎在性能上有所差异，应根据具体的应用场景选择合适的引擎。

- 数据库引擎配置：对于 PostgreSQL，建议启用并行查询、共享内存、共享缓冲池等配置。对于 MySQL，建议启用 InnoDB 引擎，并配置合理的 innodb_buffer_pool_size 和 innodb_log_file_size。
- 硬件配置：PostGIS 的性能与硬件配置密切相关。建议使用 SSD 存储，并配置足够的内存。对于 CPU，建议使用多核处理器，并启用超线程技术。
- 索引优化：PostGIS 支持多种索引类型，如 B-tree、GiST、SP-GiST 等。应根据具体的数据类型和查询需求选择合适的索引类型。同时，应定期维护索引，确保其处于最佳状态。

有关数据库引擎配置和硬件优化的详细建议，请参考 Section 13.11 和 Section 13.4。

4.3.4 常见问题 FAQ

1. 如何安装 PostGIS？

PostGIS 的安装方法取决于所使用的数据库引擎和操作系统。对于 PostgreSQL，可以通过包管理器（如 apt、yum）或从源码编译安装。对于 MySQL，则需要通过 MySQL 的官方安装程序进行安装。安装完成后，需要执行一些初始化脚本，以启用 PostGIS 功能。

2. 如何配置 PostGIS？

PostGIS 的配置主要涉及数据库引擎的配置和 PostGIS 自身的配置。对于 PostgreSQL，需要在 postgresql.conf 和 pg_hba.conf 中进行配置。对于 MySQL，则需要修改 my.cnf 文件。PostGIS 的配置包括启用 PostGIS 功能、设置搜索路径等。

3. 如何优化 PostGIS 查询？

PostGIS 查询的优化方法包括：使用合适的索引类型（如 GiST 索引）、使用合适的查询语句（如使用 ST_Contains 而不是 ST_ContainsStrict）、使用合适的数据类型（如使用 geometry 而不是 geography）等。此外，还可以通过调整数据库引擎的配置来优化查询性能。

4. 如何验证 PostGIS 数据？

PostGIS 提供了多种验证数据的方法，如 ST_IsValid、ST_IsValidReason 等。这些函数可以用于验证几何数据的合法性，如是否存在自相交、是否存在空值等。此外，还可以通过可视化的方式验证数据，如使用 GIS 软件进行地图显示。

4.4 Geometry Validation

PostGIS 是符合开放地理空间联盟（OGC）简单要素规范（Simple Features specification）的。该规范定义了几何数据的基本概念，如 *simple* 和 *valid*。这些定义允许我们验证几何数据的合法性。

Simple Features geometry model to represent spatial objects in a consistent and unambiguous way that supports efficient computation. (Note: the OGC SF and SQL/MM have the same definitions for simple and valid.)

4.4.1 Simple Geometry

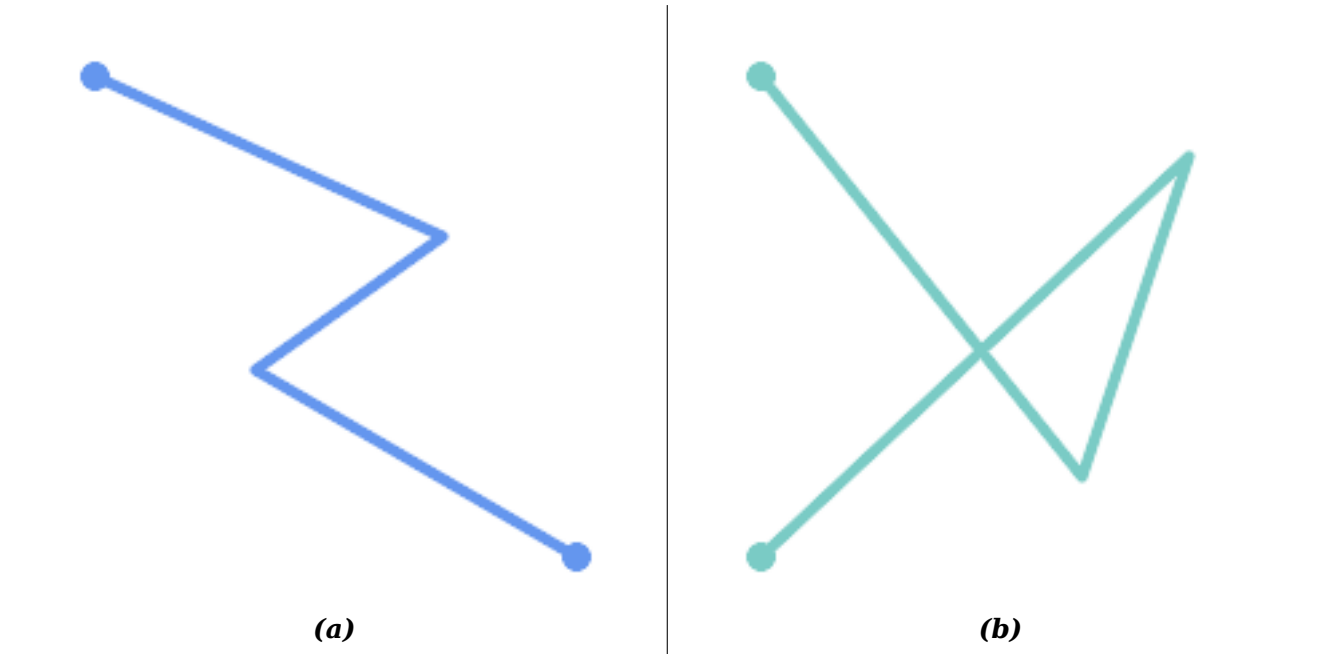
A *simple* geometry is one that has no anomalous geometric points, such as self intersection or self tangency.

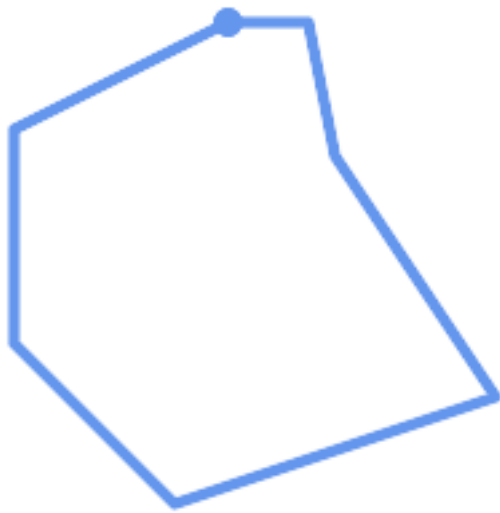
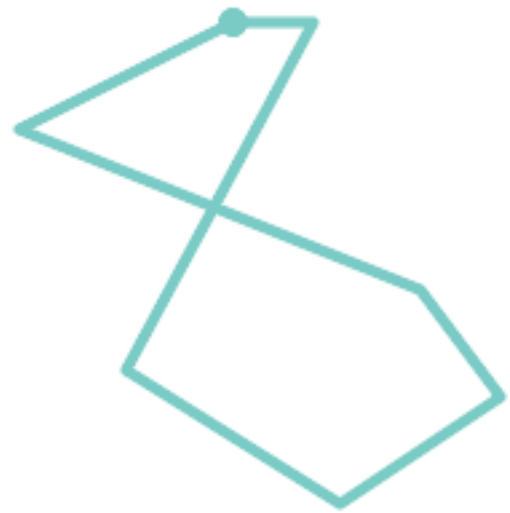
POINT 0 0000000000000000 000000.

MULTIPOINT 000000 (POINT) 000000 (0000000000000000) 000000.

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

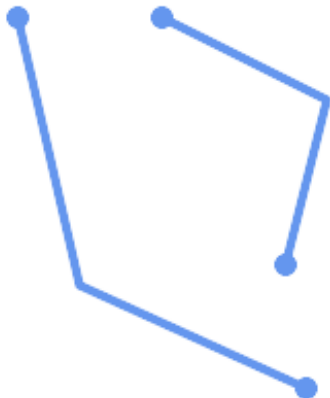
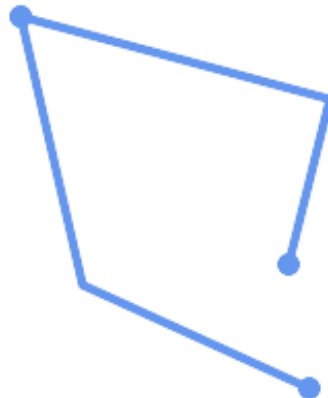
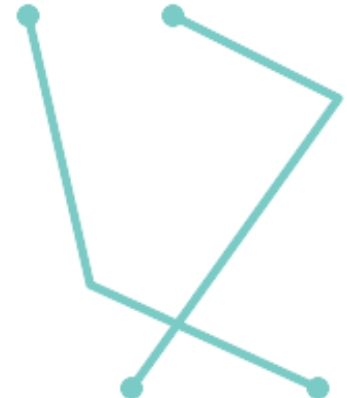
(a) and **(c)** are simple LINESTRINGs. **(b)** and **(d)** are not simple. **(c)** is a closed Linear Ring.



**(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](#).

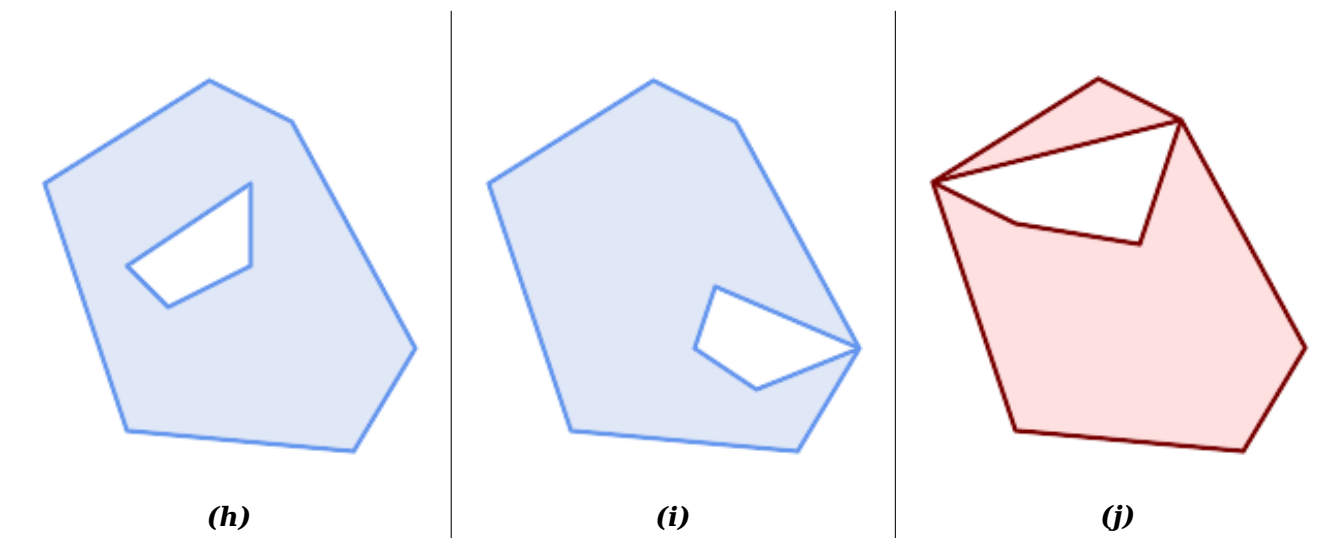
4.4.2 Valid Geometry

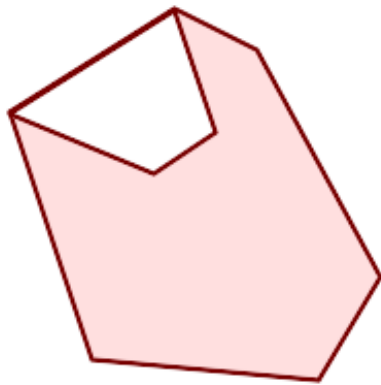
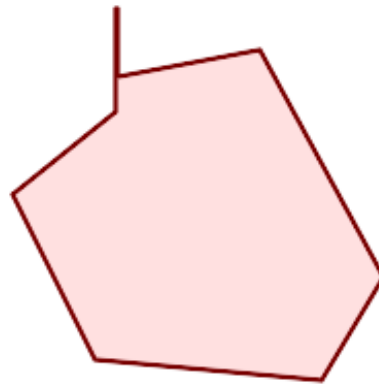
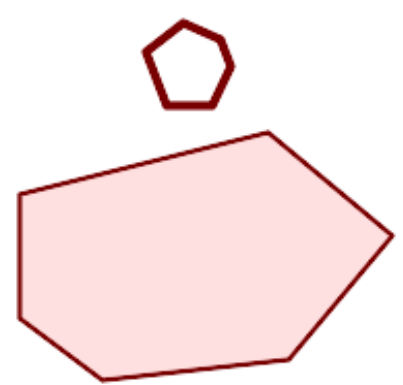
Geometry validity primarily applies to 2-dimensional geometries (POLYGONs and MULTIPOLYGONs) . Validity is defined by rules that allow polygonal geometry to model planar areas unambiguously.

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.

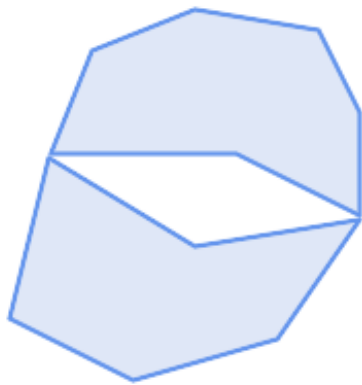
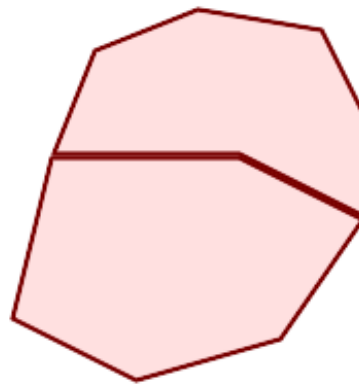
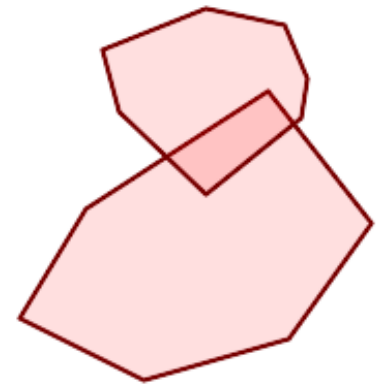


**(k)****(l)****(m)**

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 表

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 表结构:

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

表字段:

srid 唯一标识符 (SRS) 的 ID。

auth_name 标识 SRS 的权威名称。通常为“EPSG”或“PROJ4”。
AUTH_NAME 列。

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) 描述 SRS。


```

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

```

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

proj4text PostGIS 存储 proj4 字符串。 PROJ4TEXT 存储 SRID 存储 proj4 字符串。 示例：

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

示例 <http://trac.osgeo.org/proj/> 存储 proj4 字符串。 spatial_ref_sys.sql 存储 EPSG 存储 SRTEXT 存储 PROJ4TEXT 字符串。

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

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

4.5.2 SPATIAL_REF_SYS 字符串

PostGIS 的 SPATIAL_REF_SYS 存储 proj 字符串，存储 3000 个字符串，存储 proj4 字符串。 存储 4326 - WGS 84 Long Lat, 4269 - NAD 83 Long Lat, 3395 - WGS 84 World Mercator, 2163 - US National Atlas Equal Area, 存储 NAD 83 与 WGS 84 UTM (带; zone) 字符串。 存储 UTM 字符串, 6 个字符串。

存储 SPATIAL_REF_SYS 字符串 <http://spatialreference.org/> 字符串。

存储 4326 - WGS 84 Long Lat, 4269 - NAD 83 Long Lat, 3395 - WGS 84 World Mercator, 2163 - US National Atlas Equal Area, 存储 NAD 83 与 WGS 84 UTM (带; zone) 字符串。 存储 UTM 字符串, 6 个字符串。

存储 (字符串) 字符串。 存储 SPATIAL_REF_SYS 字符串, 存储 ESRI 字符串 spatialreference.org 字符串。

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 数据类型

4.6.1 几何数据类型

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**:

- **几何类型修饰符**。POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON。Z, M 或 ZM 表示 3D 几何体。'LINESTRINGM' 表示 3D 几何体，'POINTZM' 表示 3D 几何体。
- 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 特征规格 (Simple Features Specification for SQL)” 定义了 GIS 数据, 包括表、视图、索引、函数、操作符、数据类型等。OpenGIS 规范定义了 OpenGIS 数据库系统必须实现的功能。

```
\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)  |
```

该视图包含以下信息:

f_table_catalog, f_table_schema, f_table_name 指定了表或视图的完整名称。" f_table_catalog " 指定了数据库名称。" f_table_schema " 指定了模式名称。" f_table_name " 指定了表或视图名称。PostgreSQL 数据库使用 " public " 模式。

f_geometry_column 指定了几何列的名称。

coord_dimension 指定了坐标维数 (2, 3, 或 4 维)。

srid 指定了空间参考 ID, SPATIAL_REF_SYS 表 (foreign key)。

type 指定了数据类型。数据类型包括: POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION, XYM, POINTM, LINESTRINGM, POLYGONM, MULTIPOINTM, MULTILINESTRINGM, MULTIPOLYGONM, GEOMETRYCOLLECTIONM。数据类型 "GEOMETRY" 是无效的。

4.6.3 geometry_columns 表

AddGeometryColumn() 函数用于向数据库中添加新的几何列, SQL 语句 (bulk insert) 插入。该函数使用 geometry_columns 表来记录新列的信息。PostGIS 2.0 版本中, typmod 参数用于指定数据类型。

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

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

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

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

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

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

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

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

创建表并注册几何列。使用 `typmod` 参数。
`geometry_columns` 表将记录几何列的元数据。使用 `typmod` 参数。
`geometry_columns` 表将记录几何列的元数据。

```
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 命令:

```
\d pois_ny;
```

创建表并注册几何列。使用 `typmod` 参数。

Column	Type	Modifiers
gid	integer	not null default nextval('pois_ny_gid_seq'::regclass)
poi_name	text	
cat	character varying(20)	
geom	geometry(Point,4326)	
geom_2160	geometry	

Indexes:

"pois_ny_pkey" PRIMARY KEY, btree (gid)

Check constraints:

"enforce_dims_geom_2160" CHECK (st_ndims(geom_2160) = 2)

"enforce_geotype_geom_2160" CHECK (geometrytype(geom_2160) = 'POINT'::text
OR geom_2160 IS NULL)

"enforce_srid_geom_2160" CHECK (st_srid(geom_2160) = 2160)

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 脚本文件

PostGIS 支持使用格式化的 SQL 脚本文件。Oracle 风格的 SQL 脚本文件，SQL 脚本文件 SQL “INSERT” 脚本文件 (piping) 脚本文件。

脚本文件 (roads.sql) 脚本文件:

```
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 脚本文件 PostgreSQL 脚本文件。

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

4.7.2 shp2pgsql: ESRI shapefile 脚本文件

shp2pgsql 脚本文件 ESRI shapefile 文件，PostGIS/PostgreSQL 脚本文件 SQL 脚本文件。脚本文件 (command line) 脚本文件。

shp2pgsql 脚本文件，PostGIS 脚本文件 shp2pgsql-gui 脚本文件。shp2pgsql-gui 脚本文件 pgAdmin III 脚本文件。

c|a|d|p -- 脚本文件:

- c 脚本文件 shapefile 脚本文件。脚本文件。
- a 脚本文件 shapefile 脚本文件。脚本文件。
- d 脚本文件 (drop) 脚本文件 shapefile 脚本文件。
- p 脚本文件 SQL 脚本文件，脚本文件。脚本文件。

-? 脚本文件。

-D 脚本文件 PostgreSQL “ 脚本文件 (dump)” 脚本文件。脚本文件 -a, -c 脚本文件 脚本文件。脚本文件 “ 脚本文件” SQL 脚本文件。脚本文件。

-s [**<FROM_SRID>:**]**<SRID>** 脚本文件 SRID 脚本文件。脚本文件 shapefile 脚本文件 FROM_SRID 脚本文件。脚本文件 SRID 脚本文件。FROM_SRID 脚本文件 -D 脚本文件。

-k 脚本文件 (脚本文件, 脚本文件) 脚本文件。shapefile 脚本文件。

```
-i DBF 64 bigint 32 64
  bigint.
```

-I ☒☒☒☒ GiST ☒☒☒☒☒☒☒☒.

[illegible]

```
COLUMNNAME DBFFIELD1
AVERYLONGCOLUMNNAME DBFFIELD2
```

-S $\square\square$ (multi) $\square\square\square\square\square\square\square\square\square\square\square\square$. $\square\square\square\square\square\square\square\square\square\square\square\square\square\square$ ($\square$: $\square\square\square\square\square\square\square\square\square\square$
 $\square\square\square\square\square\square\square\square\square\square\square\square\square\square$) $\square\square\square\square\square\square\square\square\square\square$.

[illegible][illegible]

-e [REDACTED]. [REDACTED]
[REDACTED]. " [REDACTED]" -D [REDACTED]
[REDACTED].

```
-W <encoding> [encoding] (DBF [encoding]) [encoding]. [encoding], DBF [encoding]
[encoding] UTF8 [encoding]. [encoding] SQL [encoding] SET CLIENT_ENCODING to UTF8 [encoding]
[encoding], [encoding] UTF8 [encoding]
[encoding].
```

```
-N <policy> NULL [options] -- insert*([options]), skip([options]), abort([options])
```

```
-n DBF [REDACTED]. [REDACTED] shapefile [REDACTED], [REDACTED] DBF
[REDACTED]. [REDACTED] shapefile [REDACTED]
```

-G (XXXX/XXXXXXXXXX) XXXXX WGS84 XXXX (SRID=4326) XXXXXXXXXX.

```
-T <tablename> [REDACTED]. -X [REDACTED]
[REDACTED]. PostgreSQL [REDACTED]
[REDACTED].
```

```
-X <tablespace> [tablespace_name] [primary key] [column_name]
[constraint_name] [-I] [index_name] [index_type] [index_options]
```

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

[illegible]

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

UNIX `pipe` `pipe`:

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


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。其用法如下：

```
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 指定是否导出原始数据。如果指定为 raw，则导出原始数据；否则，导出经过 remap 处理的数据。
- m filename 指定 remap 文件的名称。默认值为 10。该文件用于指定 remap 规则。PostgreSQL 支持 B-Tree、R-Tree、GiST 索引。

4.9 索引

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，特别是多边形数据。GiST 索引是一种通用的索引方法，适用于各种类型的数据。

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) 是一种通用的索引方法，适用于各种类型的数据。PostGIS 使用 GiST 索引来存储和查询空间数据。R-Tree 索引适用于空间数据，特别是多边形数据。
- **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 index stores the bounding box of the geometry (the *envelope*) and the pointer to the actual geometry. The *envelope* is the bounding box of the geometry.

GIS indexes are implemented as B-Trees. The index stores the bounding box of the geometry (the *envelope*) and the pointer to the actual geometry. The *envelope* is the bounding box of the geometry.

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

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

PostGIS 2D indexes are implemented as GiST. The index stores the bounding box of the geometry (the *envelope*) and the pointer to the actual geometry. The *envelope* is the bounding box of the geometry.

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

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

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

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

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

4.9.2 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 index stores the bounding box of the geometry (the *envelope*) and the pointer to the actual geometry. The *envelope* is the bounding box of the geometry.

A BRIN index stores the bounding box enclosing all the geometries contained in the rows in a contiguous set of table blocks, called a *block range*. When executing a query using the index the block ranges are scanned to find the ones that intersect the query extent. This is efficient only if the data is physically ordered so that the bounding boxes for block ranges have minimal overlap (and ideally are

mutually exclusive). The resulting index is very small in size, but is typically less performant for read than a GiST index over the same data.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

To summarize using BRIN for spatial data:

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

4.9.3 GiST 索引

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

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

GIS 索引支持多种数据类型，包括点、线、面、多面体、几何集合等。在 PostgreSQL 中，GIS 索引使用 SP-GiST 技术实现。SP-GiST 索引是一种分区搜索树，它可以将搜索空间划分为大小不等的分区。除了空间索引，SP-GiST 还用于加速许多其他类型的数据搜索，如电话路由、IP 路由、子串搜索等。对于更多信息，请参阅 [PostgreSQL 手册](#)。

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

对于 2D 几何数据类型，PostGIS 2.0 引入了 `n` 索引，用于支持多面体索引：

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

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

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

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

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

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

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

There is no support for kNN searches at the moment.

4.9.4 故障排除

PostgreSQL 的 GiST 索引支持多种数据类型。但是，并非所有数据类型都支持所有操作符。在创建索引之前，请确保您的数据类型和操作符都支持。PostgreSQL 的 GiST 索引支持多种数据类型，但并非所有数据类型都支持所有操作符。在创建索引之前，请确保您的数据类型和操作符都支持。

以下是一些常见的故障排除步骤（仅供参考）：

- 检查查询计划并确认您的查询实际上计算了您需要的内容。错误的 JOIN，无论是忘记还是连接到错误的表，都可能意外地多次检索表记录。要获取查询计划，请在查询前执行 EXPLAIN。
- 确保已收集有关表中值的数量和分布的统计信息，以便为查询规划器提供更多信息，以便做出有关索引使用的决策。VACUUM ANALYZE 将同时计算两者。

您应该定期真空您的数据库。许多 PostgreSQL DBAs 运行 VACUUM 作为非高峰期的 cron 作业，定期进行。

- 有时，您可以通过设置 **SET ENABLE_SEQSCAN=OFF** 来强制使用索引。但是，这可能会导致性能下降。B-Tree 索引通常比 GiST 索引更快。如果您遇到性能问题，请尝试启用 ENABLE_SEQSCAN。
- 有时，您可以通过设置 (cost) 参数来调整查询规划器的行为。在 postgresql.conf 中，您可以设置 random_page_cost。默认值为 4，您可以将其设置为 1 或 2 以提高性能。
- 如果 **SET ENABLE_SEQSCAN TO OFF;** 没有帮助，您的查询可能使用了 SQL 构造，这是 Postgres 规划器目前无法优化的。您可能需要重写查询，使其更容易处理。例如，带有内联 SELECT 的子查询可能无法产生高效的计划，但可能可以通过使用 LATERAL JOIN 来重写。

对于更多信息，请参阅 Postgres 手册中的 [Query Planning](#) 部分。

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 $\{0, 1, 2, F\}$:

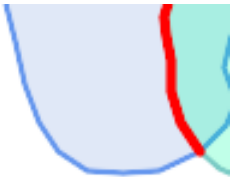
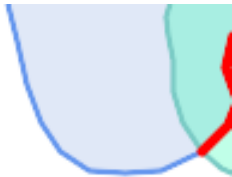
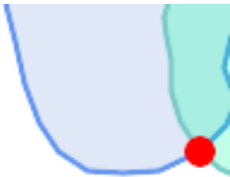
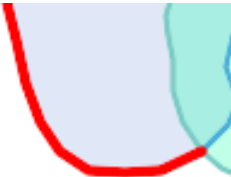
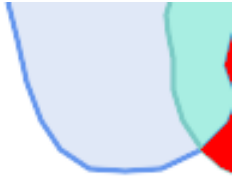
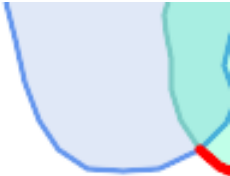
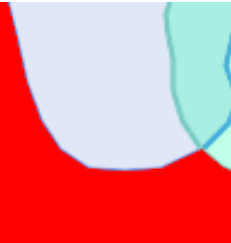
- 0 => point
- 1 => line
- 2 => area
- F => empty set

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

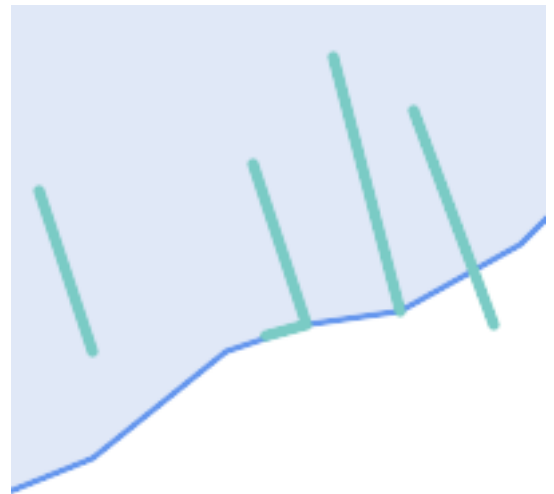
In some cases the named spatial relationships are insufficient to provide a desired spatial filter condition.



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';
```

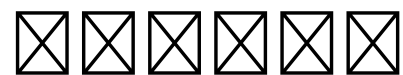
```
kilometers
-----
4.89151904172838
```

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

Chapter 6



6.1

6.1.1 ☐ ☐ ☐ ☐ ☐

[illegible]

XXXXXXXXXXXXXXXXXXXXXXX, (XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX) XXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXXXXXXX, XX TOAST XXXXXXXXXXXX. XXXXXXXX
X, XXXXXXXX 80 XXXXXXXXXXXXXXXX 3 XXXXXXXX, TOAST XXXX 8,225 XXXXXXXX
X.

The authors would like to thank the anonymous reviewers for their helpful comments. This work was supported by the National Natural Science Foundation of China (Grant No. 81873001) and the National Key Research and Development Program of China (Grant No. 2018YFB0504600). The authors also thank the TOAST team for providing the TOAST software.

XXXXXXXXXXXXXXXXXXXXX, "EXPLAIN ANALYZE" PostgreSQL XXXXXXXXXXXXX. XX
XXXXXXXXXXXXXXXXXXXXX, PostgreSQL XXXXXXXXXXXXXXXXXXXXXXXXXXXX:
<http://archives.postgresql.org/pgsql-performance/2005-02/msg00030.php>

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

6.1.2 ☐ ☐ ☐ ☐

PostgreSQL の TOAST 領域について。詳しくは、
<https://www.postgresql.org/docs/12/astorage.html>
を参照してください。

1. 在配置文件中添加以下配置项，并设置其值为 `off`。


```
enable_seqscan TO off;
```

 2. 重启数据库。
 3. 在配置文件中添加以下配置项，并设置其值为 `on`。


```
enable_seqscan TO on;
```

 4. 重启数据库。

[illegible]

```
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 `trigger` `bbox` " " `trigger` `bbox` `UPDATE`

6.2

`CREATE CLUSTER postgres ON LOCAL DISK WITH NO LOGGING; CREATE DATABASE mydb IN CLUSTER postgres;`

1. PostgreSQL 12.1 PostGIS 3.0.0 GiST 3.11.0. 2. GiST 3.11.0 NULL 3.11.0. 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. 100. 101. 102. 103. 104. 105. 106. 107. 108. 109. 110. 111. 112. 113. 114. 115. 116. 117. 118. 119. 120. 121. 122. 123. 124. 125. 126. 127. 128. 129. 130. 131. 132. 133. 134. 135. 136. 137. 138. 139. 140. 141. 142. 143. 144. 145. 146. 147. 148. 149. 150. 151. 152. 153. 154. 155. 156. 157. 158. 159. 160. 161. 162. 163. 164. 165. 166. 167. 168. 169. 170. 171. 172. 173. 174. 175. 176. 177. 178. 179. 180. 181. 182. 183. 184. 185. 186. 187. 188. 189. 190. 191. 192. 193. 194. 195. 196. 197. 198. 199. 200. 201. 202. 203. 204. 205. 206. 207. 208. 209. 210. 211. 212. 213. 214. 215. 216. 217. 218. 219. 220. 221. 222. 223. 224. 225. 226. 227. 228. 229. 230. 231. 232. 233. 234. 235. 236. 237. 238. 239. 240. 241. 242. 243. 244. 245. 246. 247. 248. 249. 250. 251. 252. 253. 254. 255. 256. 257. 258. 259. 260. 261. 262. 263. 264. 265. 266. 267. 268. 269. 270. 271. 272. 273. 274. 275. 276. 277. 278. 279. 280. 281. 282. 283. 284. 285. 286. 287. 288. 289. 290. 291. 292. 293. 294. 295. 296. 297. 298. 299. 300. 301. 302. 303. 304. 305. 306. 307. 308. 309. 310. 311. 312. 313. 314. 315. 316. 317. 318. 319. 320. 321. 322. 323. 324. 325. 326. 327. 328. 329. 330. 331. 332. 333. 334. 335. 336. 337. 338. 339. 340. 341. 342. 343. 344. 345. 346. 347. 348. 349. 350. 351. 352. 353. 354. 355. 356. 357. 358. 359. 360. 361. 362. 363. 364. 365. 366. 367. 368. 369. 370. 371. 372. 373. 374. 375. 376. 377. 378. 379. 380. 381. 382. 383. 384. 385. 386. 387. 388. 389. 390. 391. 392. 393. 394. 395. 396. 397. 398. 399. 400. 401. 402. 403. 404. 405. 406. 407. 408. 409. 410. 411. 412. 413. 414. 415. 416. 417. 418. 419. 420. 421. 422. 423. 424. 425. 426. 427. 428. 429. 430. 431. 432. 433. 434. 435. 436. 437. 438. 439. 440. 441. 442. 443. 444. 445. 446. 447. 448. 449. 450. 451. 452. 453. 454. 455. 456. 457. 458. 459. 460. 461. 462. 463. 464. 465. 466. 467. 468. 469. 470. 471. 472. 473. 474. 475. 476. 477. 478. 479. 480. 481. 482. 483. 484. 485. 486. 487. 488. 489. 490. 491. 492. 493. 494. 495. 496. 497. 498. 499. 500. 501. 502. 503. 504. 505. 506. 507. 508. 509. 510. 511. 512. 513. 514. 515. 516. 517. 518. 519. 520. 521. 522. 523. 524. 525. 526. 527. 528. 529. 530. 531. 532. 533. 534. 535. 536. 537. 538. 539. 540. 541. 542. 543. 544. 545. 546. 547. 548. 549. 550. 551. 552. 553. 554. 555. 556. 557. 558. 559. 560. 561. 562. 563. 564. 565. 566. 567. 568. 569. 570. 571. 572. 573. 574. 575. 576. 577. 578. 579. 580. 581. 582. 583. 584. 585. 586. 587. 588. 589. 590. 591. 592. 593. 594. 595. 596. 597. 598. 599. 600. 601. 602. 603. 604. 605. 606. 607. 608. 609. 610. 611. 612. 613. 614. 615. 616. 617. 618. 619. 620. 621. 622. 623. 624. 625. 626. 627. 628. 629. 630. 631. 632. 633. 634. 635. 636. 637. 638. 639. 640. 641. 642. 643. 644. 645. 646. 647. 648. 649. 650. 651. 652. 653. 654. 655. 656. 657. 658. 659. 660. 661. 662. 663. 664. 665. 666. 667. 668. 669. 670. 671. 672. 673. 674. 675. 676. 677. 678. 679. 680. 681. 682. 683. 684. 685. 686. 687. 688. 689. 690. 691. 692. 693. 694. 695. 696. 697. 698. 699. 700. 701. 702. 703. 704. 705. 706. 707. 708. 709. 710. 711. 712. 713. 714. 715. 716. 717. 718. 719. 720. 721. 722. 723. 724. 725. 726. 727. 728. 729. 730. 731. 732. 733. 734. 735. 736. 737. 738. 739. 740. 741. 742. 743. 744. 745. 746. 747. 748. 749. 750. 751. 752. 753. 754. 755. 756. 757. 758. 759. 760. 761. 762. 763. 764. 765. 766. 767. 768. 769. 770. 771. 772. 773. 774. 775. 776. 777. 778. 779. 780. 781. 782. 783. 784. 785. 786. 787. 788. 789. 790. 791. 792. 793. 794. 795. 796. 797. 798. 799. 800. 801. 802. 803. 804. 805. 806. 807. 808. 809. 810. 811. 812. 813. 814. 815. 816. 817. 818. 819. 820. 821. 822. 823. 824. 825. 826. 827. 828. 829. 830. 831. 832.

```
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()`, `isna()` "not null" `notnull()`, `notna()`:

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

`blubb`, `geometry` ist NULL. `ALTER TABLE blubb ADD CHECK (geometry is not null);` CHECK Constraint wird erstellt.

6.3

3D 4D , 2D OpenGIS ST_AsText() ST_AsBinary() ST_Force2D() .

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

```
AddGeometryColumn('','geometry_columns', geometry_columns
geometry_columns
geometry_columns).
```

1. 删除空值: 使用 `WHERE the_geom IS NOT NULL` 来删除空值。

Chapter 7

PostGIS Reference

PostGIS is a spatial database system that provides the ability to store and retrieve geographic data. It is built on top of PostgreSQL and provides a rich set of functions for spatial analysis and manipulation.



Note
PostGIS is a spatial database system that provides the ability to store and retrieve geographic data. It is built on top of PostgreSQL and provides a rich set of functions for spatial analysis and manipulation. The Spatial Type (ST) functions are used to create and manipulate spatial data. The ST_ functions are used to perform spatial analysis and manipulation.

7.1 PostgreSQL PostGIS Geometry/Geography/Box

7.1.1 box2d

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

box3d is a type representing a 3-dimensional bounding box. It is a subtype of box2d. The ST_3DExtent function returns the 3D extent of a geometry.

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	

图例

Section 13.7

7.1.2 box3d

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

图例

box3d 数据类型在 PostgreSQL 9.5 中引入。ST_3DExtent 函数返回 box3d 数据类型。

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).

图例

图例

图例	图例
box	图例
box2d	图例
geometry	图例

图例

Section 13.7

7.1.3 geometry

geometry — geography 数据类型在 PostgreSQL 9.5 中引入。

图例

geography 数据类型在 PostgreSQL 9.5 中引入。

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

图例

图例

图例	图例
box	图例
box2d	图例
box3d	图例
bytea	图例

geography	geometry
text	geometry

几何

Section 4.1, Section 4.3

7.1.4 geometry_dump

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

几何

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.

几何

Section 13.6

7.1.5 geography

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

几何

geography 地理几何类型，用于表示具有大地测量（椭球体）坐标系统的空间特征。

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

地理几何

地理几何类型，用于表示具有大地测量（椭球体）坐标系统的空间特征。

geometry	geometry
geometry	geometry

❏

Section 4.3, Section 4.3

7.2 几何函数

7.2.1 AddGeometryColumn

AddGeometryColumn — 添加几何列。

Synopsis

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

❏

❏. schema_name ❏. srid ❏
❏ SPATIAL_REF_SYS ❏. type ❏, ❏
❏'POLYGON' ❏'MULTILINESTRING' ❏. ❏ (❏ search_path ❏
❏) ❏ SRID, ❏, ❏.

Note



❏: 2.0.0 ❏ geometry_columns ❏ geometry_columns ❏. ❏ PostgreSQL ❏. ❏ WGS84 POINT ❏
❏: ALTER TABLE some_table ADD COLUMN geom
geometry(Point,4326);
❏: 2.0.0 ❏. ❏, ❏ use_typedmod ❏, ❏
❏.

Note



❏: 2.0.0 ❏. ❏ geometry_columns ❏, ❏ typedmod ❏
❏, ❏ typedmod ❏
❏ geometry_columns ❏
❏, ❏ typedmod ❏.
Section 4.6.3 ❏.



This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**.



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



This method supports Circular Strings and Curves.

❏: 2.0.0 ❏. use_typedmod ❏. ❏ typedmod ❏
❏.

```
-- 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
                                Table "my_schema.my_spatial_table"
Column | Type | Modifiers
-----+-----+-----
id     | integer | not null default nextval('my_schema.my_spatial_table_id_seq'::regclass)

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

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

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

-- Describe the table again reveals the addition of a new geometry columns.
\d my_schema.my_spatial_table
                                addgeometrycolumn
-----+-----+-----
my_schema.my_spatial_table.geomcp_c SRID:4326 TYPE:CURVEPOLYGON DIMS:2
(1 row)
```

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 — XXXXXXXXXXXXXXXXXXXXXXXX.

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 XXXXXXXXXX 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

[illegible]

```
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
```

¶¶

ST_SRID

7.2.5 Populate_Geometry_Columns

Populate_Geometry_Columns — Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.

Synopsis

```
text Populate_Geometry_Columns(boolean use_typmod=true);
int Populate_Geometry_Columns(oid relation_oid, boolean use_typmod=true);
```

¶¶

Populate_Geometry_Columns 用于确保几何列具有适当的类型修饰符或空间约束。该函数接受两个参数：一个布尔值 use_typmod 和一个 OID relation_oid。如果 use_typmod 为 true，则函数将尝试为所有几何列添加类型修饰符。如果 use_typmod 为 false，则函数将尝试为所有几何列添加空间约束。

如果 use_typmod 为 true，则函数将尝试为所有几何列添加类型修饰符。如果 use_typmod 为 false，则函数将尝试为所有几何列添加空间约束。该函数将返回一个整数，表示成功更新的列数。如果 use_typmod 为 true，则返回的整数将大于或等于 3。如果 use_typmod 为 false，则返回的整数将小于或等于 3。

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

oid 参数是几何列的 OID。SRID 参数是空间参考系的 ID。如果 SRID 参数为 0，则函数将尝试为所有几何列添加空间约束。如果 SRID 参数不为 0，则函数将尝试为所有几何列添加类型修饰符。

oid 参数是几何列的 OID。geometry_columns 参数是几何列的表名。如果 geometry_columns 参数为 NULL，则函数将尝试为所有几何列添加空间约束。如果 geometry_columns 参数不为 NULL，则函数将尝试为指定的几何列添加空间约束。

如果 geometry_columns 参数不为 NULL，则函数将尝试为指定的几何列添加空间约束。如果 geometry_columns 参数为 NULL，则函数将尝试为所有几何列添加空间约束。该函数将返回一个整数，表示成功更新的列数。如果 use_typmod 为 true，则返回的整数将大于或等于 3。如果 use_typmod 为 false，则返回的整数将小于或等于 3。

1.4.0 版本引入。

版本 2.0.0 中，use_typmod 参数默认为 true。在 2.0.0 之前，use_typmod 参数默认为 false。

版本 2.0.0 中，geometry_columns 参数默认为 NULL。在 2.0.0 之前，geometry_columns 参数默认为 NULL。


```
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);
```

函数

`geometry_columns` 函数返回 srid 列的 SRID 值。注意：在 schema-aware postgres 安装中，使用 `current_schema()` 函数。



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



This method supports Circular Strings and Curves.

函数

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)
\.
```

使用 `UpdateGeometrySRID` 函数将 SRID 设置为 4326：

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

使用 DDL 修改表结构：

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

如果 SRID 未知（即 'unknown'），可以使用 `ST_Transform` 函数将 SRID 转换为 3857，DDL 如下：

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

函数

UpdateRasterSRID, ST_SetSRID, ST_Transform

7.3 几何构造器 (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);
```

ST_Collect

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 新增。ST_MakeLine 新增。ST_MakeLine 新增。



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



This method supports Circular Strings and Curves.

ST_Collect: XLink

Collect 2D points.

```
SELECT ST_AsText( ST_Collect( ST_GeomFromText('POINT(1 2)'),
                             ST_GeomFromText('POINT(-2 3)') ));
```

st_astext

MULTIPOINT((1 2),(-2 3))

Collect 3D points.

```
SELECT ST_AsEWKT( ST_Collect( ST_GeomFromEWKT('POINT(1 2 3)'),
                             ST_GeomFromEWKT('POINT(1 2 4)') ) );
```

st_asewkt

MULTIPOINT(1 2 3,1 2 4)

Collect curves.

```
SELECT ST_AsText( ST_Collect( 'CIRCULARSTRING(220268 150415,220227 150505,220227 150406)',
                             'CIRCULARSTRING(220227 150406,220227 150407,220227 150406)') );

          st_astext
-----
MULTICURVE(CIRCULARSTRING(220268 150415,220227 150505,220227 150406),
CIRCULARSTRING(220227 150406,220227 150407,220227 150406))
```

例: 聚合几何

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))
```

例: 聚合几何

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
```

例:

ST_Dump, ST_AsBinary

7.3.2 ST_LineFromMultiPoint

ST_LineFromMultiPoint — 从多点点集创建线。

Synopsis

geometry **ST_LineFromMultiPoint**(geometry aMultiPoint);

例:

从多点点集创建线。

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 创建多边形。

例: 2.0 指定的SRID 创建多边形 (envelope) 与SRS 相关联。

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 — 根据给定的点创建线。

Synopsis

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

概要

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 添加了对 3D 的支持，不会丢弃 z 索引。

2.0.0 添加了对 3D 的支持，不会丢弃 z 索引。

1.4.0 添加了对 3D 的支持。与 ST_MakeLine 一起使用。与 ST_MakeLine 一起使用。

用法: 创建线

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)
```

添加 3D 支持。与 BOX3D 一起使用。

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

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

用法, 与 ST_MakeLine 一起使用。

```
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)
```

ST_MakeLine: 创建线

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)
```

ST_MakeLine: 创建线

使用 GPS 点创建线，按时间顺序排列 GPS 点。GPS 点按时间顺序排列。

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

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 .

Synopsis

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


☐☐

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.

☐☐



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 经纬度坐标.

```
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
```

☐☐

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);

返回

返回类型 (shell) 返回类型。 返回类型。

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

。 使用 **ST_LineMerge** 和 **ST_Dump** 。



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

返回: 返回

。

```
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;
```

。

```
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))
```

注意: 使用 `ST_MakePolygon` 时

必须使用 `ST_IsClosed` 函数。

```
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 值使用 `ST_MakePolygon` 时，如果 NULL 值使用 `CASE` 语句。

```
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;
```

函数

[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);

返回

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

返回: 返回

```
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);
```

返回: 返回

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;
```

返回

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);

返回

返回与 ST_Point 相同。 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.

返回

```
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)
```

返回

[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);

返回

返回与 ST_Point 相同。 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.

返回

```
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 ☐☐☐☐☐☐☐. ST_MakePoint ☐☐☐☐ OGC ☐☐☐☐☐☐.

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry.

☐☐

```
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](#)

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);

☐☐

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**

此函数在 3D 模式下运行。它使用 `ST_LineMerge` 和 `ST_Dump` 函数。



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.

创建

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))
```

函数

`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);

描述

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](#).

示例: 2.0.0 使用 SRID 2.0.0.

2.1.0 使用.

示例: 使用.

```
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))
```

示例

[ST_MakeEnvelope](#)

7.3.14 ST_HexagonGrid

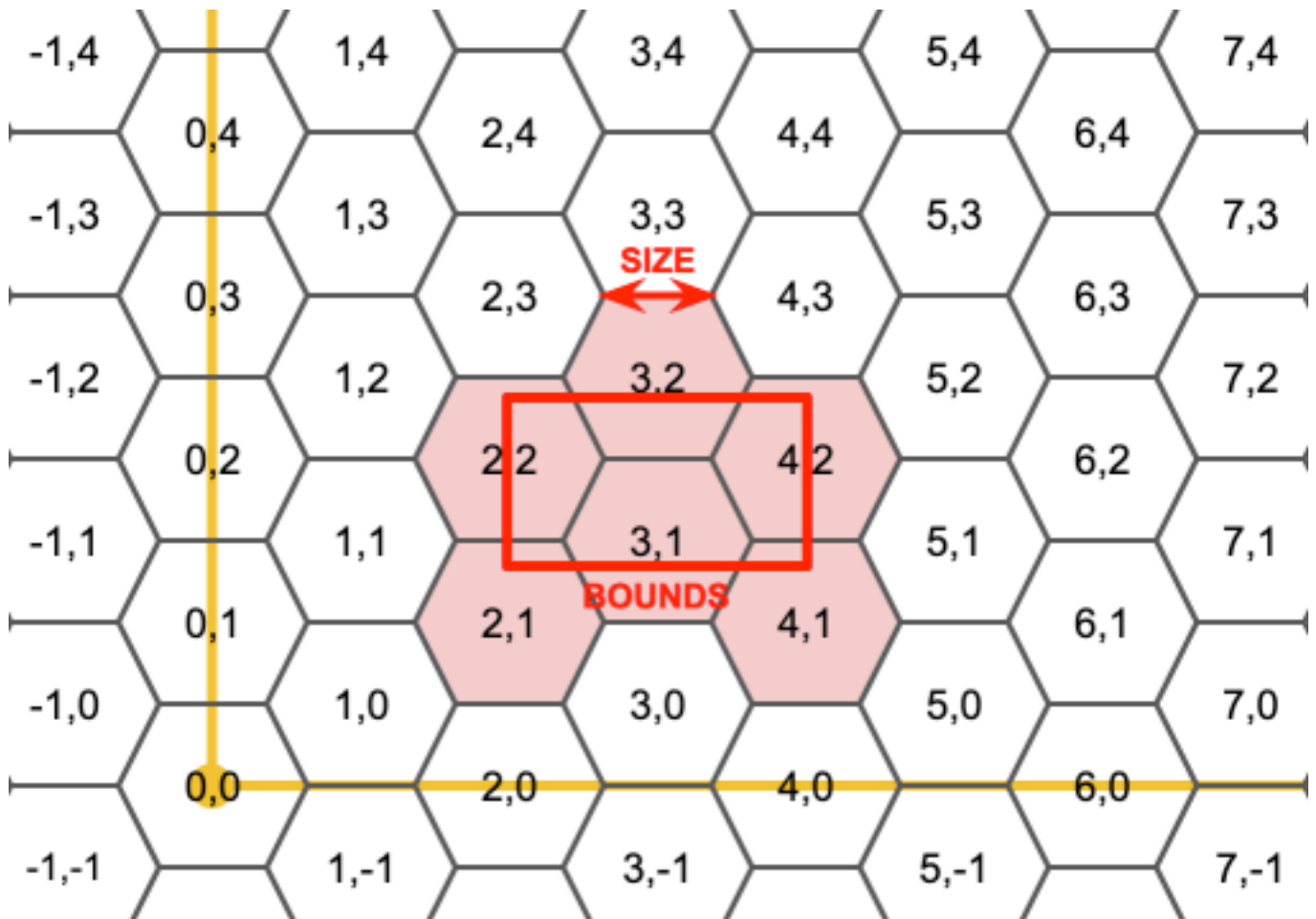
`ST_HexagonGrid` — Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.

Synopsis

setof record **ST_HexagonGrid**(float8 size, geometry bounds);

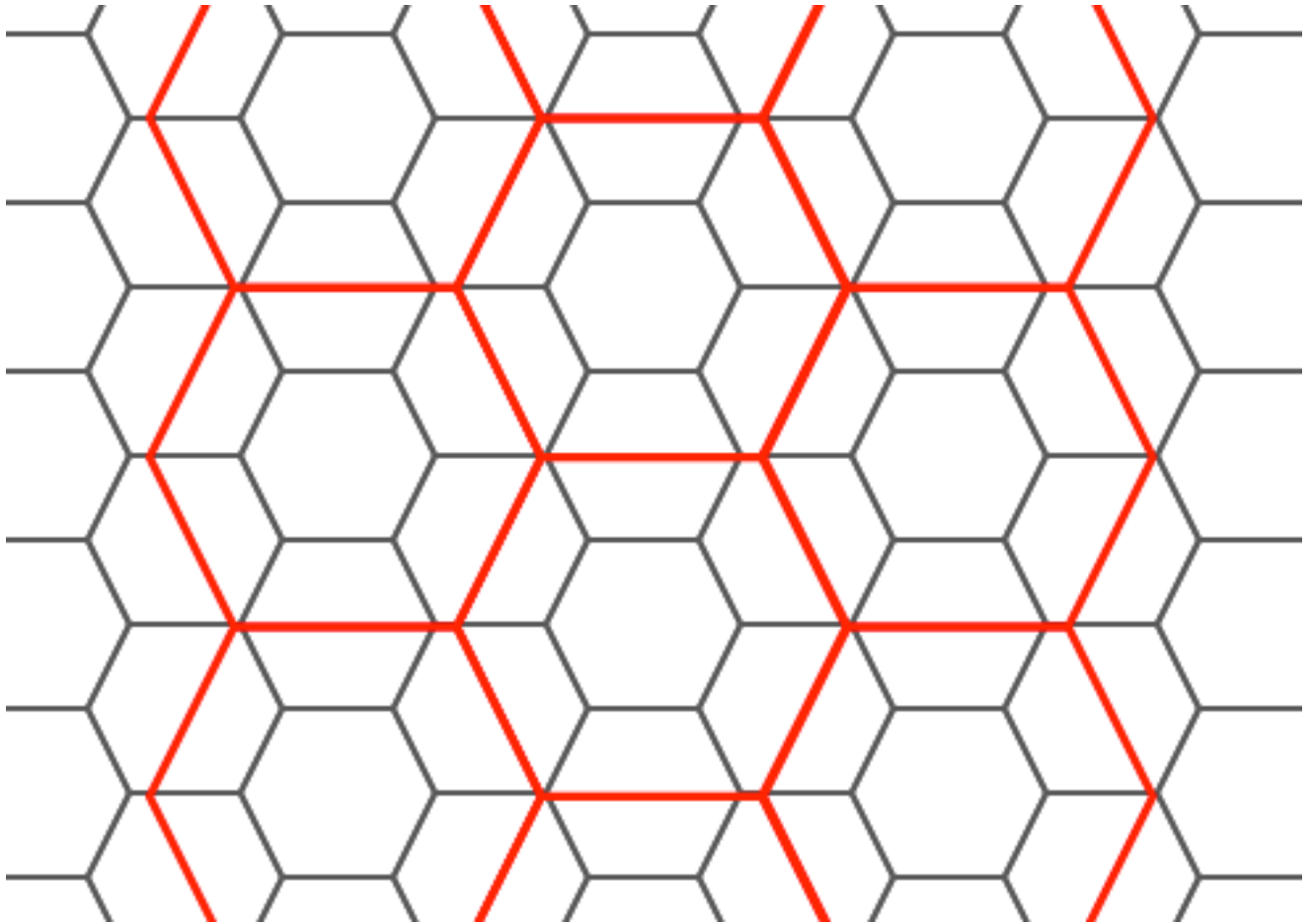
示例

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 简体中文.

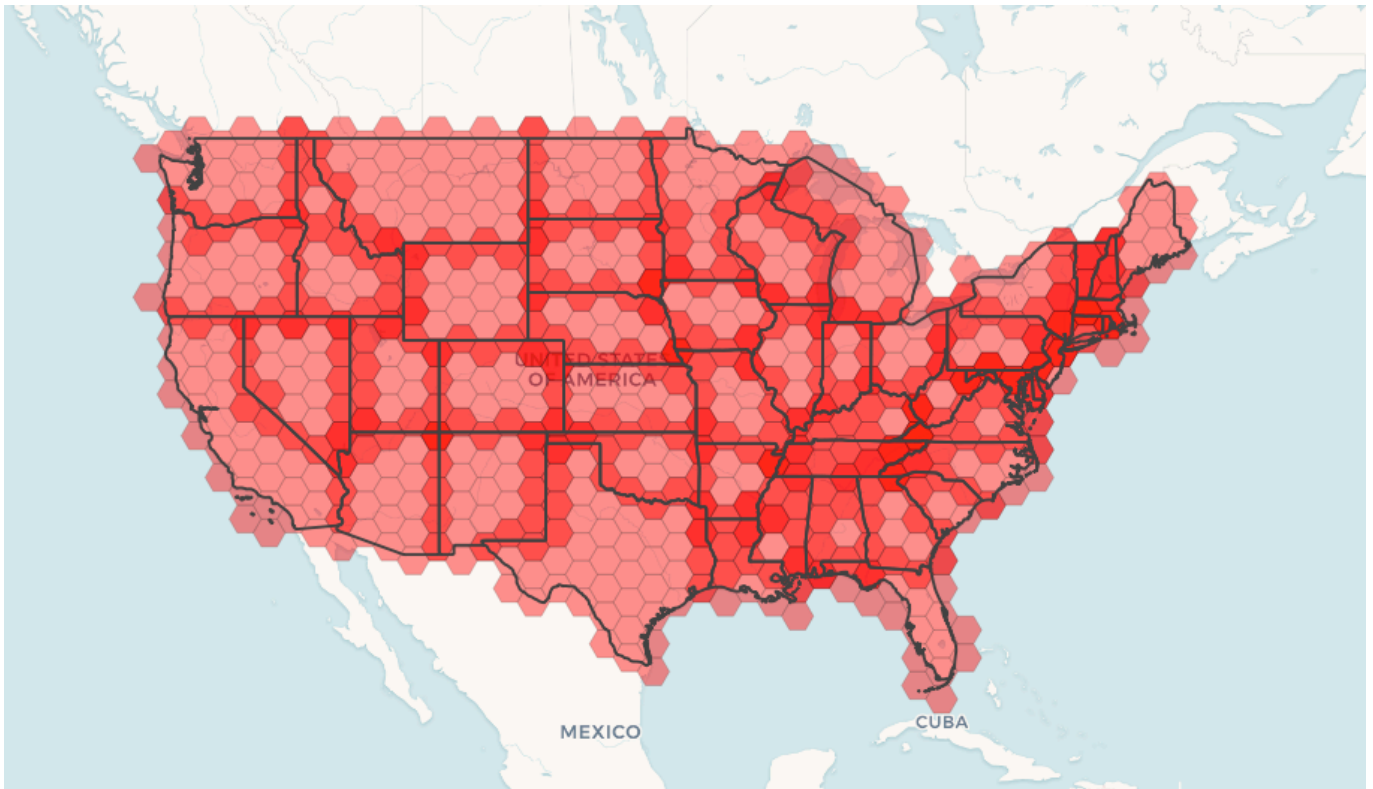
图例: 简体中文

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;
```

图例: 简体中文

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, **Tiling**(SRS, Size). This function answers the question: what grids in a given **Tiling**(SRS, Size) overlap with a given bounds.

The SRS for the output squares is the SRS provided by the bounds geometry.

Doubling or edge size of the square generates a new parent tiling that perfectly fits with the original tiling. Standard web map tilings in mercator are just powers-of-two square grids in the mercator plane.

2.1.0 ￼￼￼￼￼￼￼￼￼￼.

图例: 国家边界和网格

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
```

图例: 点分布和网格

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
```

图例: 点分布和网格

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
```

图例

[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

```
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

返回几何类型的名称。例: 'LINESTRING', 'POLYGON', 'MULTIPOINT' 等。

OGC 简单特征访问接口 (SFAI) - 简单特征访问接口 (SFAI), 简单特征访问接口 (SFAI) 的简单特征访问接口 (SFAI)。

**Note**

PostGIS 3.6.0alpha1 简体中文版

PostGIS 2.0.0 简体中文版, 支持 TIN 数据类型。

- ✓ 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).

SQL

```
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
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))) AS geom
  ) 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) 符合 OGC 3.12.3.2 的定义。ST_Boundary (位相的) 符合 OGC 3.12.2 的定义 (primitive) 返回几何对象的边界。

GEOS 支持



Note

2.0.0 版本中，GEOMETRYCOLLECTION 类型的输入将返回 NULL。2.0.0 版本中，(GEOMETRYCOLLECTION) NULL 输入将返回 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

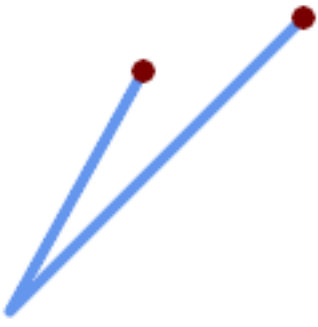
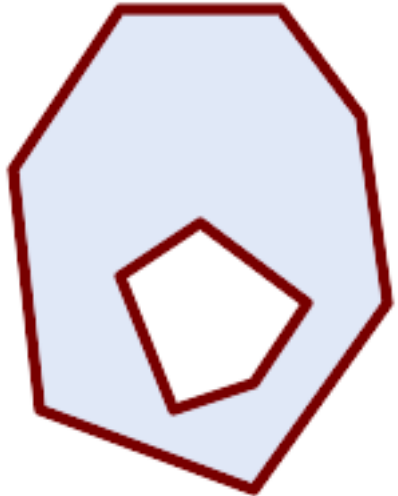


This function supports 3d and will not drop the z-index.

文档: 2.1.0 版本中，ST_Boundary 返回的几何对象将包含 z-index。

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

文档

 图例 <pre>SELECT ST_Boundary(geom) FROM (SELECT 'LINESTRING(100 150,50 60, 70 80, 160 170)::geometry As geom) As f;</pre> ST_AsText output <pre>MULTIPOINT((100 150),(160 170))</pre>	 图例 <pre>SELECT ST_Boundary(geom) FROM (SELECT 'POLYGON ((10 130, 50 190, 110 190, 140 150, 150 80, 100 10, 20 40, 10 130), (70 40, 100 50, 120 80, 80 110, 50 90, 70 40))::geometry As geom) As f;</pre> ST_AsText output <pre>MULTILINESTRING((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130), (70 40,100 50,120 80,80 110,50 90,70 40))</pre>

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

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

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

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

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

☐☐

[ST_AsText](#), [ST_ExteriorRing](#), [ST_MakePolygon](#)

7.4.3 ST_BoundingDiagonal

`ST_BoundingDiagonal` — 返回给定几何图形的最小外接矩形的对角线。

Synopsis

geometry **ST_BoundingDiagonal**(geometry geom, boolean fits=false);

☐☐

返回给定几何图形的最小外接矩形的对角线。如果 `fits` 为 `true`，则返回的对角线将穿过几何图形的质心。如果 `fits` 为 `false`，则返回的对角线将穿过矩形的中心。如果几何图形是一个点，则返回一个长度为 2 的数组，表示该点的 X 和 Y 坐标。

`fits` 参数 (best fit) 控制是否返回最佳拟合的对角线。如果 `fits` 为 `true`，则返回的对角线将穿过几何图形的质心。如果 `fits` 为 `false`，则返回的对角线将穿过矩形的中心。如果几何图形是一个点，则返回一个长度为 2 的数组，表示该点的 X 和 Y 坐标。

返回的几何图形将具有与输入相同的 SRID。



Note

对于非凸多边形，返回的对角线可能不会穿过几何图形的质心。在这种情况下，返回的对角线将穿过矩形的中心。

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 函数。

Synopsis

integer **ST_CoordDim**(geometry geomA);

返回

ST_Geometry 函数。

MM 函数, **ST_NDims** 函数。

- ✓ 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).

返回

```
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
```

返回

ST_NDims

7.4.5 ST_Dimension

ST_Dimension — ST_Geometry 函数。

Synopsis

integer **ST_Dimension**(geometry g);

POINT 0, LINESTRING 1, POLYGON 2, GEOMETRYCOLLECTION. OGC s2.1.1.1.

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.

Version: 2.0.0, TIN

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

**Note**

1.3.4 版本 (curve) 支持。1.3.4 版本支持。

- ✓ 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.

示例

```
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)
```

示例, TIN 示例

```
-- 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 0 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 ☒☒☒☒☒☒☒☒☒☒☒.

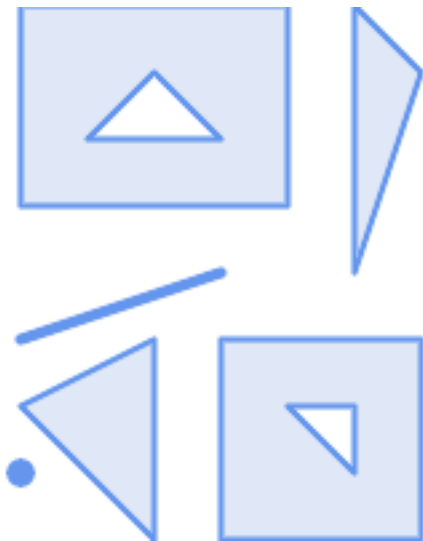
- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.

- ✔ 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)
```

图例



```
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 )),
          )
      ) As g
    ) As foo;
path | ST_AsText(geom)
-----+-----
1 | POINT(0 1)
2 | LINESTRING(0 3,3 4)
3 | POLYGON((2 0,2 3,0 2,2 0))
4 | POLYGON((3 0,3 3,6 3,6 0,3 0),(5 1,4 2,5 2,5 1))
5 | 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)

繁體中文, TIN 繁體中文

```
-- 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 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 *n*th segment start point in the LINESTRING. For a POLYGON the paths are {*i*, *j*} where *i* is the ring number (1 is outer; inner rings follow) and *j* is the segment start point position in the ring.

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)

简体中文, **TIN** 简体中文

```
-- 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.

☐☐

```
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))

```

☐☐

Box2D, Box3D, ST_OrientedEnvelope

7.4.12 ST_ExteriorRing

ST_ExteriorRing — 返回多边形的外环。

Synopsis

geometry **ST_ExteriorRing**(geometry a_polygon);

☐☐

POLYGON 类型 (exterior ring) 返回一个多边形。如果输入为 NULL 则返回 NULL。



Note

ST_ExteriorRing 返回一个多边形。ST_Dump 返回一个多边形。

- ✔ 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.

☐☐

```

--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))')
  )
)

```


提取点

```
--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);
```

多面体表面, TIN 三角网

```
-- 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.

例

```
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
```

例

[ST_CurveToLine](#), [ST_PointN](#)

7.4.16 ST_InteriorRingN

`ST_InteriorRingN` — 返回多边形中的第 `N` 个内部环。

Synopsis

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

例

返回多边形中的第 `N` 个内部环。如果 `N` 为负数，则返回外部环。如果 `N` 为 0，则返回 NULL。



Note

如果多边形是无效的，则返回 NULL。使用 `ST_Dump` 可以验证多边形是否有效。

- ✓ 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.

例

```
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;
```

例

[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**。 **POINT** (点) 永远是 **TRUE**。

Synopsis

boolean **ST_IsClosed**(geometry g);

例

LINESTRING 是否是闭合的 **TRUE** 或 **FALSE**。 **POINT**, **POLYGON** (多边形) 永远是 **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 多面体表面 (polyhedral surface) 支持。
- ✓ This function supports Polyhedral surfaces.

相关链接


```

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

```

返回

ST_IsRing

7.4.20 ST_IsCollection

ST_IsCollection — 判断几何对象是否是集合，如果是，返回 TRUE，否则返回 FALSE。

Synopsis

boolean **ST_IsCollection**(geometry g);

返回

判断几何对象是否是集合，如果是，返回 TRUE，否则返回 FALSE：

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



Note

判断几何对象是否是集合，如果是，返回 TRUE，否则返回 FALSE。



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



This method supports Circular Strings and Curves.

返回

```
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
```


例

```
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);

例

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 简体中文。



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



This function supports M coordinates.

简体中文

`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);

简体中文

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 简体中文。



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);

返回

如果几何体是简单的，则返回 TRUE。如果几何体不是简单的，则返回 FALSE。OGC 简单要素实现规范 (OGC Simple Features Implementation Specification) 中，"OpenGIS 简单要素实现规范 (Ensuring OpenGIS compliance of geometries)" 确保了 OpenGIS 的兼容性。



Note

SQL-MM 中 ST_IsSimple(NULL) 返回 0，PostGIS 中 NULL 返回 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.

返回

```
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)
```

返回

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。返回类型是 float。



Note

返回 (返回) OGC 简单特征实现规范, 提取器 (extractor) 返回类型是 float。



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 返回。

[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.

例

```
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
```

例

[ST_CoordDim](#), [ST_Dimension](#), [ST_GeomFromEWKT](#)

7.4.29 ST_NPoints

ST_NPoints — 返回几何体中的点数量 (整数)。

Synopsis

integer **ST_NPoints**(geometry g1);

例

返回几何体中的点数量。对于多面体表面，返回其顶点的数量。
 2.0.0 版本之前，对于多面体表面 (polyhedral surface) 返回其面的数量。



Note

1.3.4 版本之前，对于曲线 (curve) 返回其弧段的数量。1.3.4 版本之后，返回其顶点的数量。

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

例

```
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 — Returns the number of rings in a polygon.

Synopsis

integer **ST_NRings**(geometry geomA);

ST_NRings returns the number of rings in a polygon. NumInteriorRings returns the number of interior rings in a polygon.

- ✓ 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 — Returns the number of geometries in a collection.

Synopsis

integer **ST_NumGeometries**(geometry geom);

返回

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

更新: 2.0.0 支持多边形, 支持 TIN 数据类型。

更新: 2.0.0 支持空几何返回 NULL。2.0.0 之前, 空几何返回 1。

- ✔ 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).

示例

```
--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
```

函数

ST_GeometryN, ST_Multi

7.4.32 ST_NumInteriorRings

ST_NumInteriorRings — 返回多边形内部环的数量。

Synopsis

integer **ST_NumInteriorRings**(geometry a_polygon);

返回

返回多边形内部环的数量。空几何返回 NULL。

- ✔ This method implements the SQL/MM specification. SQL-MM 3: 8.2.5
- 更新: 2.0.0 支持多边形。

☒☒

```
--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。 返回多边形内部环的数量。

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。 返回多边形内部环的数量。 **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](#).

7.4.36 ST_PatchN

ST_PatchN — ST_Geometry 数据类型。

Synopsis

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

返回

返回 POLYHEDRALSURFACE, POLYHEDRALSURFACEM 类型 1-N 面 (面) 的几何体。如果 n 为 NULL 则返回 NULL。返回的 ST_GeometryN 数据类型。ST_GeometryN 返回的 ST_GeometryN 数据类型。



Note

返回 1-N 面。



Note

返回的 ST_GeometryN 数据类型 ST_Dump 返回的 ST_GeometryN 数据类型。

2.0.0 版本支持。



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.

返回

```
--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))
```

返回

ST_AsEWKT, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries

7.4.37 ST_PointN

`ST_PointN` — `ST_LineString` ☒☒ `ST_CircularString` ☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

Synopsis

```
geometry ST_PointN(geometry a_linestring, integer n);
```



The `getenv` function returns a pointer to the value of the environment variable `name`. If `name` is the null pointer (`0`), then `getenv` returns the pointer to the environment variable whose name is the null pointer. If `name` is not the null pointer and points to a string that is not the name of an environment variable, the behavior is undefined. If `name` is the name of an environment variable that has been set to the null character (`0`), then `getenv` returns the pointer to the null character. If `name` is the name of an environment variable that has not been set, then `getenv` returns the null pointer.



Note!

Note

0.8.0 00000000 OGC 000000 1-000000. OGC 00000000 (000000) 00000000. 00000000 0-0000000000000000.



Note!

Note

```
XXXXXXXXXXXXXXXXXXXXXXXX N XXXXXXXXXX, XXXX ST_Dump XXXXX
XXXXXXXXXX.
```

- ✔ 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!

Note

```

PostGIS: 2.0.0
PostGIS: 2.0.0
NULL
PostGIS: 2.3.0 (-1)

```



```
-- 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_LineString 与 ST_CircularString 函数支持。

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

与

ST_EndPoint, ST_PointN

7.4.40 ST_Summary

ST_Summary — 返回几何体的摘要字符串。

Synopsis

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

与

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。

摘要字符串: 2.0.0 返回摘要字符串。

摘要字符串: 2.1.0 与。返回摘要字符串的格式如下 S 返回摘要字符串。

摘要字符串: 2.2.0 返回 TIN 摘要字符串 (curve) 返回摘要字符串。

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 的 Z 索引标志。

Synopsis

smallint **ST_Zmflag**(geometry geomA);

返回

ST_Geometry 的 Z 索引标志。

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.

示例

```
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
```

参见

[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 引入。



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))”

- 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 简体中文版。

例

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)))
```

例

[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);



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))
```

文档

ST_Force3D

7.5.8 ST_Force3D

ST_Force3D — 强制 XYZ 模式。 ST_Force3DZ 类似。

Synopsis

geometry **ST_Force3D**(geometry geomA, float Zvalue = 0.0);

文档

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) 支持。

2.1.0, 2.0.x ST_Force_3D 支持。

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.

文档

```
--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))
```

文档

ST_AsEWKT, **ST_Force2D**, **ST_Force3DM**, **ST_Force3DZ**

7.5.9 ST_Force3DZ

ST_Force3DZ — 强制 XYZ 模式。

Synopsis

geometry **ST_Force3DZ**(geometry geomA, float Zvalue = 0.0);

更新

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

更新: 2.0.0 支持多面体表面 (polyhedral surface) 类型。

更新: 2.1.0 支持, 在 2.0.x 版本中 `ST_Force3DZ` 不支持。

更新: 3.1.0. 增加了支持供应非零 Z 值。

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

更新

```
--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))
```

更新

ST_AsEWKT, **ST_Force2D**, **ST_Force3DM**, **ST_Force3D**

7.5.10 ST_Force3DM

ST_Force3DM — 将 XYM 几何体强制转换为 3D。

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 — 将几何集合强制转换为集合几何。

Synopsis

geometry **ST_ForceCollection**(geometry geomA);

❏

将几何集合强制转换为集合几何。 返回 WKB 几何集合。

版本: 2.0.0 引入 (polyhedral surface) 支持。

1.2.2 引入。 1.3.4 引入 (curve) 支持。 1.3.4 引入。

版本: 2.1.0 引入, 在 2.0.x 版本中 ST_Force_Collection 支持。



This function supports Polyhedral surfaces.



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



This method supports Circular Strings and Curves.

❏

```
-- 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 1 0,1 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

ST ForceCurve — $\text{float}[8], \text{float}[16]$ (upcast) float .

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

`CompoundCurve`, `Multisurface`. `CompoundCurve` (compoundcurve) `CC`, `Multisurface` (`Multisurface`) `MS`. `CC` and `MS` are both subclasses of `Curve`.

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);

返回

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 简体中文。

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

返回

ST_ForcePolygonCCW , **ST_IsPolygonCCW** , **ST_IsPolygonCW**

7.5.16 ST_ForceSFS

ST_ForceSFS — 简体中文 SFS 1.1 简体中文。

Synopsis

geometry **ST_ForceSFS**(geometry geomA);
 geometry **ST_ForceSFS**(geometry geomA, text version);

返回

- ✔ 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 — 简体中文 (orientation) 简体中文 (Right-Hand Rule) 简体中文。

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;

curvedastext                                     non_curvedastext
-----|-----
CURVEPOLYGON(CIRCULARSTRING(4 3,3.12132034355964 0.878679656440359, | POLYGON((4
3,3.94235584120969 2.41472903395162,3.77163859753386 1.85194970290473,
1 0,-1.12132034355965 5.12132034355963,4 3)) | 3.49440883690764
1.33328930094119,3.12132034355964 0.878679656440359, | 2.66671069905881
0.505591163092366,2.14805029 | 0.228361402466141,
```

```

| 1.58527096604839 ↵
| 0.0576441587903094,1 ↵
| 0,
| 0.414729033951621 ↵
| 0.0576441587903077,-0.1480502 ↵
| 0.228361402466137,
| -0.666710699058802 ↵
| 0.505591163092361,-1.12132034 ↵
| 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,
| 0.9999999999999991 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 π (180 degrees)
- West is azimuth $3\pi/2$ (270 degrees)

Negative azimuth values and values greater than 2π (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);



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 .



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



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);
```



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);



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).

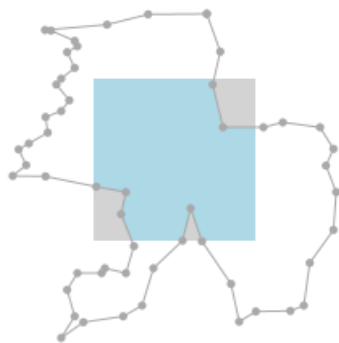
If `cartesian_hint` is set to `true`, the algorithm applies additional optimizations involving cartesian math to further reduce the resulting point number. Please note that using this option might introduce rendering artifacts if the resulting coordinates are projected into another (non-cartesian) coordinate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



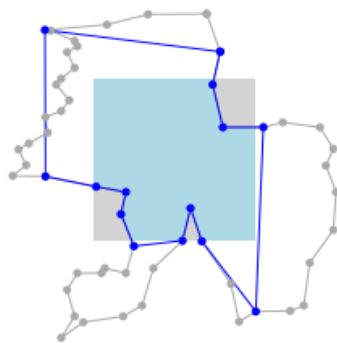
Warning

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`.

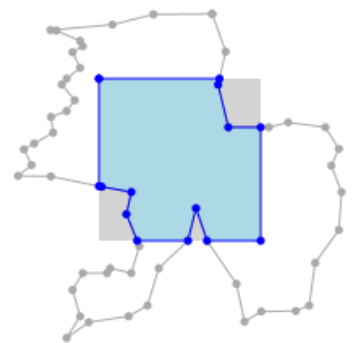
original
55 points



`ST_RemoveIrrelevantPointsForView(geom, bbox)`
15 points

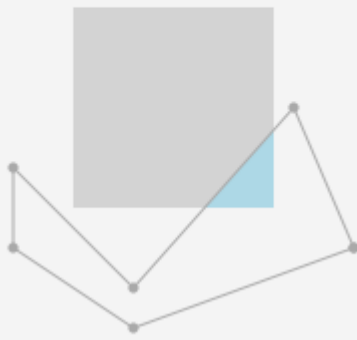


`ST_ClipByBox2D(geom, bbox)`
15 points

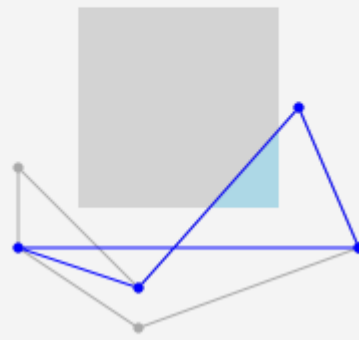


Example: `ST_RemoveIrrelevantPointsForView()` applied to a polygon. Blue points remain, the rendering result (light-blue area) within the grey view box remains as well.

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

测试

```
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))
```

测试

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);



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.
-

original
20 points, 4 parts



ST_RemoveSmallParts(geom, 30,30)
15 points, 3 parts



ST_RemoveSmallParts(geom, 50,50)
10 points, 2 parts



Example: ST_RemoveSmallParts() applied to a multi-polygon. Blue parts remain.

Availability: 3.5.0

例

```
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 — 返回几何对象的反向几何。

Synopsis

geometry **ST_Reverse**(geometry g1);

新新

新新新新新新新新新新, 新新新新新新新新新新新新新新新新.

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.

新新

```
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);

新新

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 地理几何分段.

Enhanced: 3.0.0 Segmentize geometry now produces equal-length subsegments

Enhanced: 2.3.0 Segmentize geography now produces equal-length subsegments

修复: 2.1.0 修复地理几何分段函数.

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)`)

几何

Segmentizing a line. Long segments are split evenly, and short segments are not split.

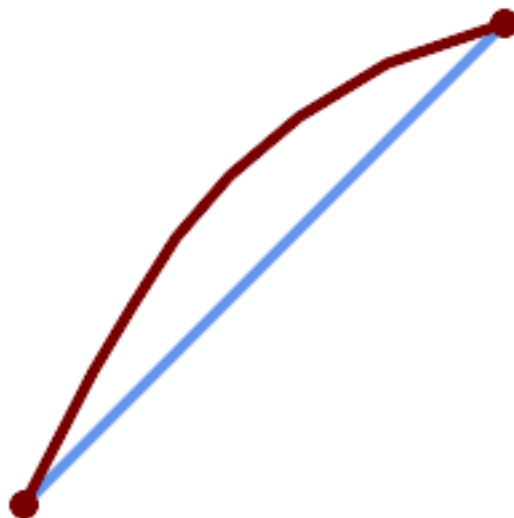
```
SELECT ST_AsText(ST_Segmentize(
  'MULTILINESTRING((0 0, 0 1, 0 9),(1 10, 1 18))'::geometry,
    5 ) );
-----
MULTILINESTRING((0 0,0 1,0 5,0 9),(1 10,1 14,1 18))
```

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

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);

返回

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 版本中，该函数在 PostgreSQL 11.0 之前版本中不可用。1.3.4 版本中，该函数在 PostgreSQL 11.0 之前版本中不可用。



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

版本: 2.0.0 版本 (polyhedral surface) 和 TIN 版本。

版本: 2.2.0 版本, 该函数名为“ST_Shift_Longitude”。



This function supports Polyhedral surfaces.



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

返回

```
--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 坐标轴上的坐标值进行包裹。

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 2.3.0 及以上版本中可用。

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 — 将几何体 (geom) 对齐到指定的精度网格 (snap)。

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);
```

注意

选项 1, 2, 3: 将几何体对齐到指定的精度网格 (cell) 指定的精度网格 (snap) 指定的精度网格。指定的精度网格指定的精度网格, 指定的精度网格指定的精度网格指定的精度网格 NULL 指定的精度网格。指定的精度网格指定的精度网格指定的精度网格。指定的精度网格指定的精度网格。

选项 4: 1.1.0 指定的精度网格指定的精度网格。指定的精度网格指定的精度网格 (指定的精度网格, 指定的精度网格) 指定的精度网格指定的精度网格指定的精度网格。指定的精度网格指定的精度网格指定的精度网格, 指定的精度网格 0 指定的精度网格指定的精度网格。



Note

指定的精度网格指定的精度网格指定的精度网格 (ST_IsSimple 指定的精度网格)。



Note

1.1.0 指定的精度网格指定的精度网格 2 指定的精度网格指定的精度网格。1.1.0 指定的精度网格指定的精度网格, 指定的精度网格指定的精度网格指定的精度网格, 指定的精度网格指定的精度网格指定的精度网格指定的精度网格。指定的精度网格指定的精度网格指定的精度网格指定的精度网格。

1.0.0RC1 指定的精度网格指定的精度网格。

1.1.0 指定的精度网格 Z 指定的精度网格 M 指定的精度网格指定的精度网格。



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

注意

```
--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 1.1111112)'),
ST_GeomFromEWKT('POINT(1.12 2.22 3.2 4.4444)'),
0.1, 0.1, 0.1, 0.01) );

                                st_asewkt
-----
LINESTRING(-1.08 2.12 2.3 1.1144,4.12 3.22 3.1 1.1144,-1.08 2.12 2.3 1.1144)

--With a 4d geometry - the ST_SnapToGrid(geom,size) only touches x and y coords but keeps m ←
and z the same
SELECT ST_AsEWKT(ST_SnapToGrid(ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 3 2.3456,
        4.111111 3.2374897 3.1234 1.1111)'),
        0.01) );

                                st_asewkt
-----
LINESTRING(-1.11 2.12 3 2.3456,4.11 3.24 3.1234 1.1111)

```


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]

The above discussion shows that the proposed method is effective in reducing the number of iterations required for convergence. The proposed method is compared with the existing methods in the literature. The proposed method is compared with the existing methods in the literature. The proposed method is compared with the existing methods in the literature.



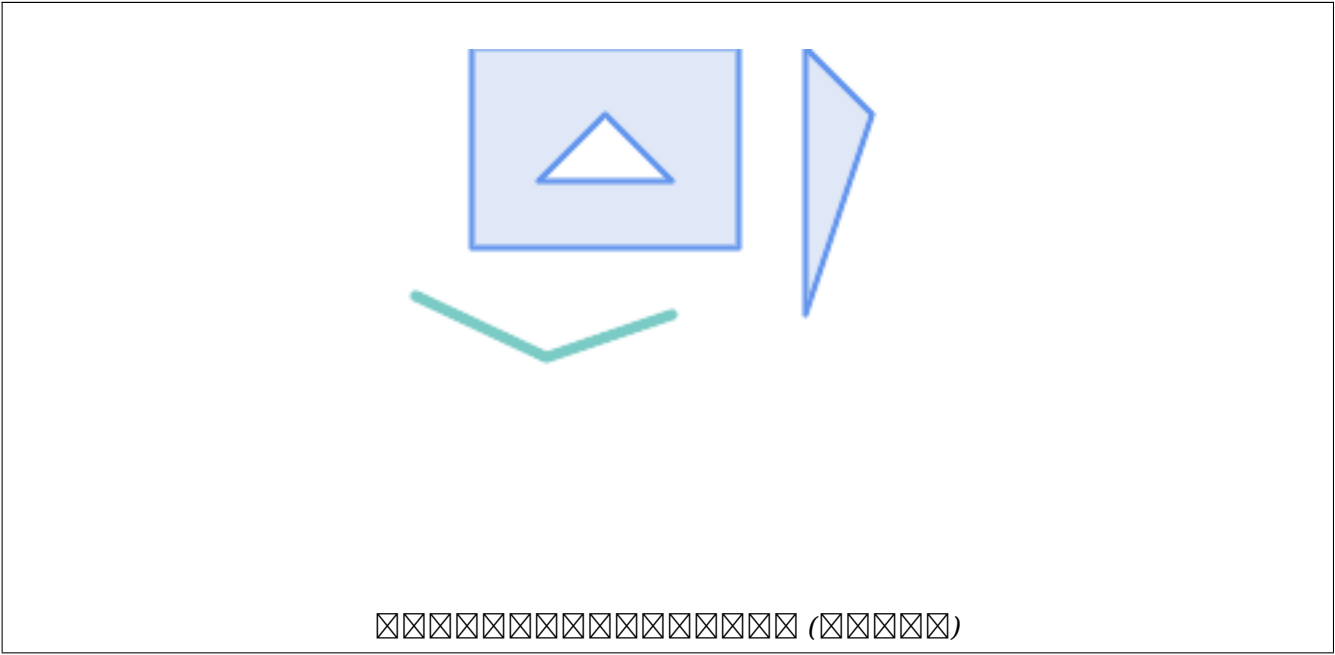
Note

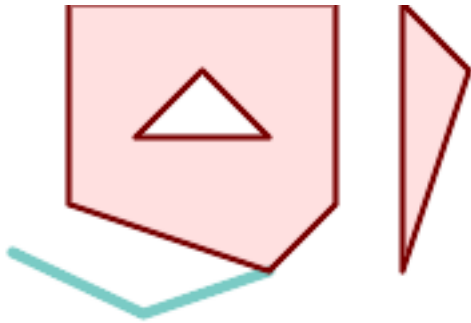
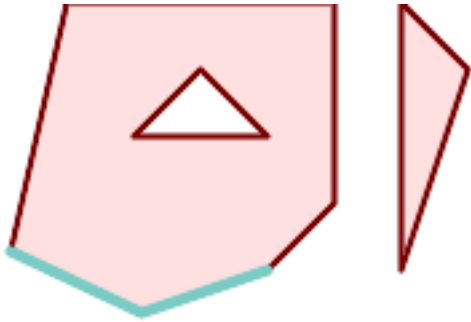
(ST IsSimple XX) (ST IsValid XX)

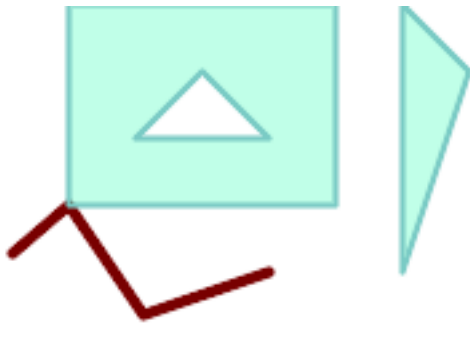
GEOS ☒ ☒ ☒ ☒ ☒

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.





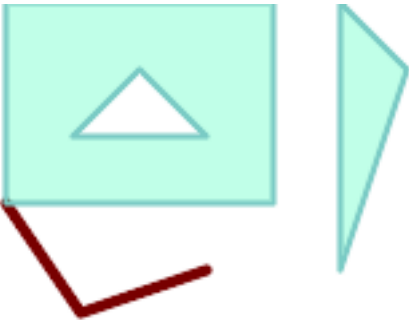
	
距離 1.01 從多邊形邊界到線段的最短距離。 距離 1.25 從多邊形邊界到線段的最短距離。	距離 1.25 從多邊形邊界到線段的最短距離。 距離 1.25 從多邊形邊界到線段的最短距離。
<pre>SELECT ST_AsText(ST_Snap(poly,line, ST_Distance(poly,line)*1.01)) AS polysnapped FROM (SELECT ST_GeomFromText('MULTIPOLYGON(((26 125, 26 200, 126 200, 126 125, 26 125), (51 150, 101 150, 76 175, 51 150)), ((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>
polysnapped	polysnapped
<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 将红线条移动到最近的青色多边形。结果是一个新的红线条。

```
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 将红线条移动到最近的青色多边形。结果是一个新的红线条。

```
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 — 交换几何体中的两个有序坐标。

Synopsis

geometry **ST_SwapOrdinates**(geometry geom, cstring ords);

函数

函数返回一个文本字符串。

ords 返回一个文本字符串，其长度为 2 乘以 M 值。 返回的字符串 x, y, z, M m 格式。
2.2.0 版本支持。

- ✓ 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).

函数

```
-- 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)
```

函数

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

ST_IsValidDetail

```
--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.

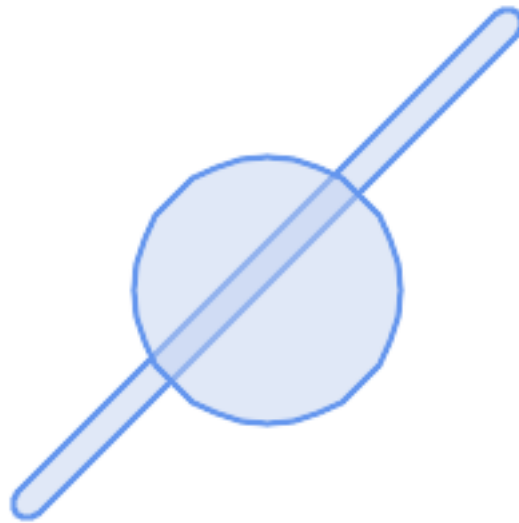
Enhanced: 3.1.0, added removal of Coordinates with NaN values.

Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS >= 3.10.0.

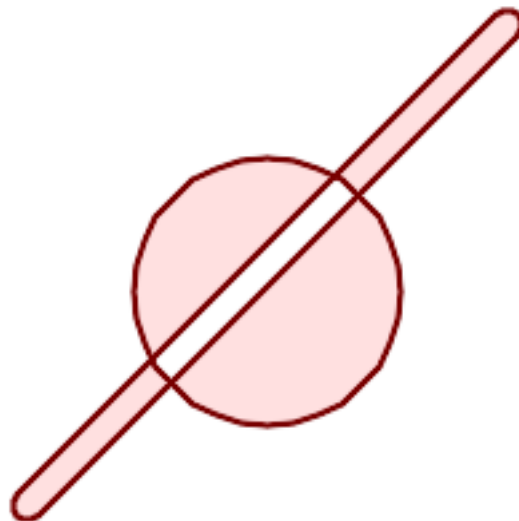


This function supports 3d and will not drop the z-index.

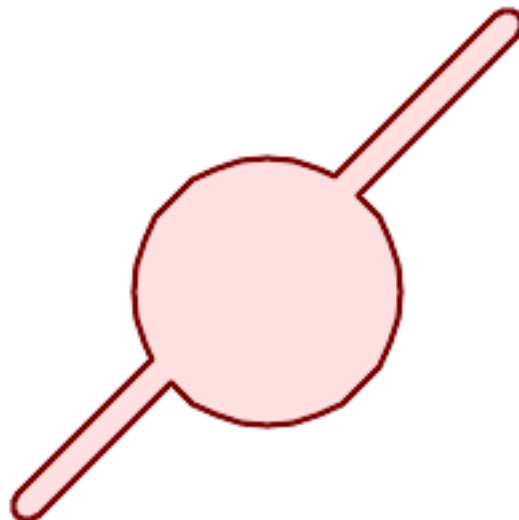




before_geom: MULTIPOLYGON of 2 overlapping polygons

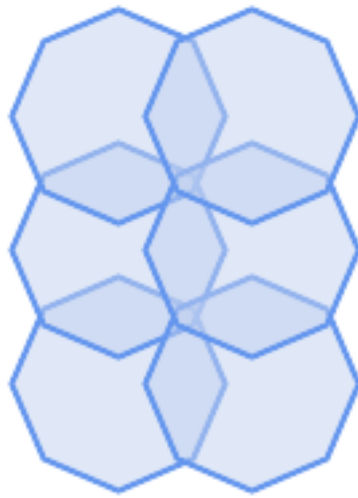


after_geom: MULTIPOLYGON of 4 non-overlapping polygons

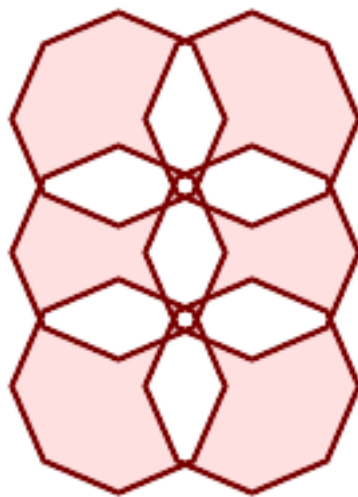


after_geom_structure: MULTIPOLYGON of 1 non-overlapping polygon

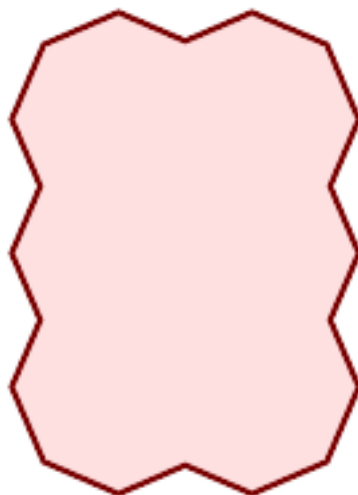
```
SELECT f.geom AS before_geom, ST_MakeValid(f.geom) AS after_geom, ST_MakeValid(f.geom, ↵
    'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,
```

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 ↵
```

 例

```
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
```

例

[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);

例

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  (curve) . 1.3.4 .

: 2.0.0  (polyhedral surface) .

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 48.99999999999999)
(1 row)
```

GDA2020 example.

```
-- using ST_Transform with automatic selection of a conversion pipeline.
SELECT ST_AsText(ST_Transform('SRID=4939;POINT(143.0 -37.0)::geometry, 7844)) AS ←
    gda2020_auto;

                gda2020_auto
-----
POINT(143.00000635638918 -36.999986706128176)
(1 row)

-- using a defined conversion (EPSG:8447)
SELECT ST_AsText(ST_TransformPipeline('SRID=4939;POINT(143.0 -37.0)::geometry,
    'urn:ogc:def:coordinateOperation:EPSG::8447')) AS gda2020_code;

                gda2020_code
-----
POINT(143.0000063280214 -36.999986718287545)
(1 row)

-- using a PROJ pipeline definition matching EPSG:8447, as returned from
-- 'projinfo -s EPSG:4939 -t EPSG:7844'.
```

```
-- NOTE: any 'axiswap' steps must be removed.
SELECT ST_AsText(ST_TransformPipeline('SRID=4939;POINT(143.0 -37.0)::geometry,
'+proj=pipeline
+step +proj=unitconvert +xy_in=deg +xy_out=rad
+step +proj=hgridshift +grids=au_icsm_GDA94_GDA2020_conformal_and_distortion.tif
+step +proj=unitconvert +xy_in=rad +xy_out=deg')) AS gda2020_pipeline;

                gda2020_pipeline
-----
POINT(143.0000063280214 -36.999986718287545)
(1 row)
```

☒☒

[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

WKT 多边形文本。



Note

WKT 多边形文本。如果文本表示的是多边形，则使用 [ST_BdPolyFromText](#) 函数。如果文本表示的是面，则使用 [ST_BuildArea](#) 函数。



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) 地理坐标制多边形文本转换为地理坐标制多边形几何体。

Synopsis

geography **ST_GeogFromText**(text EWKT);

例

WKT 点 WKT 点. SRID 4326
ST_GeographyFromText 点. 点.

例

```
--- 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)'));
```

例

ST_AsText, ST_GeographyFromText

7.8.1.4 ST_GeographyFromText

ST_GeographyFromText — WKT (例) 点.

Synopsis

geography **ST_GeographyFromText**(text EWKT);

例

WKT 点. SRID 4326 点.

例

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 PostgreSQL ST_Geometry 几何.



Note

EWKT 几何 OGC 几何, SRID(几何) 几何 PostgreSQL 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.dddddd

```
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 转 ST_Geometry

Synopsis

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

注意

OGC WKT(Well-Known Text) 转 PostGIS ST_Geometry



Note

ST_GeomFromText 支持 2 维几何, 指定 SRID 值。如果 SRID 未指定 (SRID=0) 则默认为 0。指定 SRID 值时, SRID 必须是非负整数。



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). 3.2.6.2 - 指定 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 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

警告: PostGIS 2.0.0 之前, ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') 会报错。PostGIS 2.0.0 之后, SQL/MM 标准, ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') 会报错。

注意

```
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。



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 参数 (conformance suite) 是可选的。

WKT 字符串为 null 时返回 null。

**Note**

WKT 字符串必须有效，否则将返回 null。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

例

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

例

[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);
```

例

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

OGC 3.2.6.2 - SRID (conformance suite) .

WKT null .



Note

WKT , .
 ST_GeomFromText .



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

例

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

例

[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);
```

几何

Makes a MultiPolygon from WKT with the given SRID. If SRID is not given, it defaults to 0.

OGC 3.2.6.2 - 几何 SRID 几何 (conformance suite) 几何.

WKT 几何.



Note

几何 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. SQL-MM 3: 9.6.4

几何

```
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);
```

几何

ST_GeomFromText, ST_SRID

7.8.1.14 ST_PointFromText

ST_PointFromText — 几何 SRID 几何 WKT 几何. SRID 几何, 几何 0 几何.

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 接受 2 个参数, 第一个 SRID 指定要使用的 SRID。如果 SRID 未指定, 则使用默认值 0。第二个参数是 WKT 点表示法。如果 WKT 不是有效的点表示法, 则返回 null。如果 WKT 完全无效, 则抛出错误。



Note

WKT 点表示法与 ST_GeomFromText 类似, 但 ST_GeomFromText 接受 SRID 参数。ST_PointFromText 是 OGC 3.2.6.2 - SRID 规范 (conformance suite) 的一部分。ST_MakePoint 是 OGC 3.2.6.2 - SRID 规范 (conformance suite) 的一部分。ST_Point 是 OGC 3.2.6.2 - SRID 规范 (conformance suite) 的一部分。



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). ☐ 3.2.6.2 - SRID 规范 (conformance suite) 的一部分。



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 - SRID 规范 (conformance suite) 的一部分。

**Note**

WKT 是 OGC Simple Features Implementation Specification for SQL 1.1. 的 ST_GeomFromText 的输入。



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

SQL

```
SELECT ST_PolygonFromText('POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 ↵
    42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 ↵
    42.3902909739571))');
st_polygonfromtext
-----
010300000001000000050000006...
```

```
SELECT ST_PolygonFromText('POINT(1 2)') IS NULL as point_is_notpoly;
point_is_not_poly
-----
t
```

SQL

ST_GeomFromText

7.8.1.16 ST_WKTToSQL

ST_WKTToSQL — WKT(Well-Known Text) 转换为 ST_Geometry。与 ST_GeomFromText 类似。

Synopsis

geometry **ST_WKTToSQL**(text WKT);

SQL



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

SQL

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);

几何

OGC EWKB(Extended Well-Known Binary) 几何 PostGIS ST_Geometry 几何。



Note

EWKB 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).

几何

NAD83 几何 (SRID 4269) 几何 LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932) 几何



Note

几何: 几何 (\\) 几何 (') 几何, standard_conforming_strings 几何 \ " 几何. 几何 AsEWKB 几何。

```
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')
```

几何

ST_AsBinary, ST_AsEWKB, ST_GeomFromWKB

7.8.2.3 ST_GeomFromWKB

ST_GeomFromWKB — WKB(Well-Known Binary) 转 SRID 的函数。

Synopsis

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

返回

ST_GeomFromWKB 将 WKB SRID(可选 ID) 转换为 geometry。 使用 SQL (Geometry Factory) 函数。 使用 ST_WKBToSQL 函数。

SRID 可选, 默认为 0(unknown)。

✓ 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.

返回

```
--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~\\111\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\111\\300)\\111\\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)
```

返回

[ST_WKBToSQL](#), [ST_AsBinary](#), [ST_GeomFromEWKB](#)

7.8.2.4 ST_LineFromWKB

ST_LineFromWKB — 指定 SRID 的 WKB 数据转换为 LINESTRING 类型。

Synopsis

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

注意

ST_LineFromWKB 将 WKB 数据转换为 SRID(指定的 ID) 的 LINESTRING 类型。 - 指定 SRID, LINESTRING 类型 - 指定 SRID。 使用 SQL Geometry Factory 函数。 SRID 指定为 0 表示未指定。 指定 bytea 类型, NULL 表示空。



Note

OGC 3.2.6.2 - 指定 SRID 的 WKB 数据 (conformance suite) 转换为 LINESTRING。



Note

指定 LINESTRING 类型, 使用 ST_GeomFromWKB 函数。 指定 ST_GeomFromWKB 函数, 指定 LINESTRING 类型。



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

注意

```
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
```

注意

ST_GeomFromWKB, ST_LinestringFromWKB

7.8.2.5 ST_LinestringFromWKB

ST_LinestringFromWKB — 指定 SRID 的 WKB 数据转换为 LINESTRING 类型。

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);

返回

ST_PointFromWKB 将 WKB 转换为 SRID(指定 ID) 的 POINT 几何体 - POINT 几何体。返回 SQL 几何体 (Geometry Factory) 格式。

SRID 指定, 默认为 0。返回 bytea 格式, NULL 表示失败。

- ✓ 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.

返回

```
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)
```

返回

ST_GeomFromWKB, ST_LineFromWKB

7.8.2.7 ST_WKBToSQL

ST_WKBToSQL — WKB(Well-Known Binary) 格式 ST_Geometry 几何体。返回 SRID 指定 ST_GeomFromWKB 格式。

Synopsis

geometry **ST_WKBToSQL**(bytea WKB);

⚠️

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

⚠️

ST_GeomFromWKB

7.8.3 Other Formats

7.8.3.1 ST_Box2dFromGeoHash

ST_Box2dFromGeoHash — GeoHash 字符串 BOX2D 字符串。

Synopsis

box2d **ST_Box2dFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

⚠️

GeoHash 字符串 BOX2D 字符串。

If no precision is specified ST_Box2dFromGeoHash returns a BOX2D based on full precision of the input GeoHash string.

precision 字符串, ST_Box2dFromGeoHash 将 GeoHash 字符串转换为 BOX2D 字符串。返回的 BOX2D 字符串格式为: BOX2D(xmin ymin xmax ymax)。

2.1.0 字符串。

⚠️

```
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)
```


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 OGC GML 3.2.1 或 3.1.1 格式。

支持的 OGC GML 版本包括：

- GML 3.2.1 格式
- GML 3.1.1 格式 (SF-2 (GML 3.1.0 到 3.0.0 格式))
- 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: 2.0.0 版本 (polyhedral surface) 和 TIN 几何对象。

支持: 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) 可以是 2D 或 3D 几何对象。PostGIS 支持 Z 索引的 ST_GeomFromGML 函数将 2D 几何对象转换为 3D 几何对象。

GML 几何对象 SRS 指定。PostGIS 支持 SRID，因此 ST_GeomFromGML 函数支持 SRS 指定。GML 几何对象的 srsName 属性指定，返回的几何对象将具有相同的 SRS。

ST_GeomFromGML 函数将 GML 几何对象转换为 PostGIS 几何对象。该 GML 几何对象可以是 XLink 几何对象。



Note

ST_GeomFromGML 函数 SQL/MM 几何对象。

例: srsName 指定

```
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 指定

```
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: 多面体

```
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)))

```

☒☒

Section [2.2.3](#), [ST_AsGML](#), [ST_GMLToSQL](#)

7.8.3.4 ST_GeomFromGeoJSON

ST_GeomFromGeoJSON — GeoJSON 格式数据转换为 PostGIS 几何类型。

Synopsis

```

geometry ST_GeomFromGeoJSON(text geomjson);
geometry ST_GeomFromGeoJSON(json geomjson);
geometry ST_GeomFromGeoJSON(jsonb geomjson);

```

注意

GeoJSON 格式在 PostGIS 中支持。

ST_GeomFromGML 从 JSON 格式 (geometry fragment) 创建几何。JSON 格式

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 支持 JSON-C 0.9 格式。



Note

JSON-C 格式在 PostGIS 中支持。JSON-C 格式



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

注意

```
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)
```

注意

ST_AsText, ST_AsGeoJSON, Section 2.2.3

7.8.3.5 ST_GeomFromKML

ST_GeomFromKML — 从 KML 格式创建 PostGIS 几何。

Synopsis

geometry **ST_GeomFromKML**(text geomkml);

OGC KML ☐☐☐☐☐☐☐ PostGIS ST_Geometry ☐☐☐☐☐☐☐☐.

OGC KML 文档:

- KML 2.2.0 ☒☒☒☒☒☒

OGC KML ☒☒: <http://www.opengeospatial.org/standards/kml>

Availability: 1.5, requires lxml2 2.6+



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



ST_GeomFromKML [XXX] SQL/MM [XXXXXXXXXXXX].

	☒☒: srsName ☒☒☒☒☒☒☒



Section 2.2.3, ST AsKML

7.8.3.6 ST_GeomFromTWKB

ST GeomFromTWKB — TWKB("Tiny Well-Known Binary") .

Synopsis

```
geometry ST_GeomFromTWKB(bytea twkb);
```


ST_GeomFromTWKB(  TWKB("Tiny Well-Known Binary")   (WKB)                       



```
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 `[[[...]]]` ST_Geometry `[[...]]`. `[[...]]` ST_GeomFromGML `[[...]]`
`[[...]]`.

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 (XXXXXX)
1.5 XXXXX. LibXML2 1.6 XXXXX.

Version: 2.0.0 (polyhedral surface) TIN

Version: 2.0.0 SRID 0.

Section 2.2.3, ST_GeomFromGML, ST_AsGML

7.8.3.8 ST_LineFromEncodedPolyline

ST LineFromEncodedPolyline —  (polyline)                             .

Synopsis

```
geometry ST_LineFromEncodedPolyline(text polyline, integer precision=5);
```



```

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 — ￼￼￼ WKT(Well-Known Text) ￼￼￼ 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);
```

￼￼

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 ￼￼￼￼￼, ￼￼￼, ￼￼￼ 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).

**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
F03F0000000000000F03F000000000000F03F00000000000F03
F0000000000000000000000000000000000000000000000');
```

```
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);
```

返回

Returns the OGC/ISO **Well-Known Binary** (WKB) representation of the geometry. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding, either little-endian ('NDR') or big-endian ('XDR').

WKB format is useful to read geometry data from the database and maintaining full numeric precision. This avoids the precision rounding that can happen with text formats such as WKT.

To perform the inverse conversion of WKB to PostGIS geometry use **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.

2.0.0 支持, 支持 TIN 支持.

2.0.0 支持.

2.0.0 支持.

1.5.0 支持.

2.0.0 支持. 支持. ST_AsBinary('POINT(1 2)') 返回, n st_asbinary(unknown) is not unique error. ST_AsBinary('POINT(1 2)::geometry'); 返回. legacy.sql 支持.



This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**.



This method implements the SQL/MM specification. SQL-MM 3: 5.1.37



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.


```
SELECT ST_Ashexewkb(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
      which gives same answer as

SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326)::text;

st_ashexewkb
-----
0103000020E6100000010000000500
00000000000000000000000000000000
000000000000000000000000000000F03F
000000000000F03F000000000000F03F000000000000F03
F000000000000000000000000000000000000000000000000
```

7.9.3 Other Formats

7.9.3.1 ST_AsEncodedPolyline

ST_AsEncodedPolyline — 将几何体编码为 Polyline 格式。

Synopsis

text **ST_AsEncodedPolyline**(geometry geom, integer precision=5);

返回

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 版本引入。

返回

返回

```
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`@
```

ST_AsEncodedPolyline (segmentize) 将几何体编码为 Polyline 格式，并指定分段大小。

```
-- 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;
```

返回 \$ 符号后的字符串。


```

<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.

The `options` argument can be used to add BBOX or CRS in GeoJSON output:

- 0: means no option
- 1: GeoJSON BBOX
- 2: GeoJSON Short CRS (￼: EPSG:4326)
- 4: GeoJSON Long CRS (￼: urn:ogc:def:crs:EPSG::4326)
- 8: GeoJSON Short CRS if not EPSG:4326 (default)

The `id_column` parameter is used to set the "id" member of the returned GeoJSON features. As per GeoJSON RFC, this SHOULD be used whenever a feature has a commonly used identifier, such as a primary key. When not specified, the produced features will not get an "id" member and any columns other than the geometry, including any potential keys, will just end up inside the feature's "properties" member.

The GeoJSON specification states that polygons are oriented using the Right-Hand Rule, and some clients require this orientation. This can be ensured by using `ST_ForcePolygonCCW`. The specification also requires that geometry be in the WGS84 coordinate system (SRID = 4326). If necessary geometry can be projected into WGS84 using `ST_Transform`: `ST_Transform( geom, 4326 )`.

GeoJSON can be tested and viewed online at geojson.io and geojsonlint.com. It is widely supported by web mapping frameworks:

- [OpenLayers GeoJSON Example](#)
- [Leaflet GeoJSON Example](#)
- [Mapbox GL GeoJSON Example](#)

1.3.4 ￼￼￼￼￼￼￼￼￼.

1.5.0 ￼￼￼￼￼￼￼￼￼.

￼￼￼: 2.0.0 ￼￼￼￼￼￼￼￼ (default arg) ￼￼￼￼￼￼ (named arg) ￼￼￼.

Changed: 3.0.0 support records as input

Changed: 3.0.0 output SRID if not EPSG:4326.

Changed: 3.5.0 allow specifying the column containing the feature id



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

￼￼

Generate a FeatureCollection:

```
SELECT json_build_object(
    'type', 'FeatureCollection',
    'features', json_agg(ST_AsGeoJSON(t.*, id_column =
> 'id')::json)
)
FROM ( VALUES (1, 'one', 'POINT(1 1)::geometry),
               (2, 'two', 'POINT(2 2)'),
               (3, 'three', 'POINT(3 3)')
       ) as t(id, name, geom);
```

```
{ "type" : "FeatureCollection", "features" : [ { "type": "Feature", "geometry": { "type": "Point", "coordinates": [1,1] }, "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);
```

返回

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 2.1.2 版本, GML 3 3.1.1 版本.

' 选项 ' (bitfield) 选项. CRS 选项 GML 选项, 选项/选项 选项.

- 0: GML Short CRS (选项: EPSG:4326), 选项
- 1: GML Long CRS (选项: urn:ogc:def:crs:EPSG::4326)
- 2: GML 3 选项, 选项 srsDimension 选项.
- 4: GML 3 选项, 选项 <Curve> 选项 <LineString> 选项.
- 16: 选项/选项 (选项: srid=4326) 选项. 选项. 选项 (axis order) 选项, GML 3.1.1 选项. 选项 选项, 选项.
- 32: 选项 (envelope) 选项.

选项 (选项) 选项' 选项 选项' 选项. 选项 NULL 选项'gml' 选项.

1.3.2 选项.

1.5.0 选项.

选项: 2.0.0 选项. 选项 GML 3 选项'4' 选项. GML 3 选项 TIN 选项. 选项'32' 选项.

选项: 2.0.0 选项 (named arg) 选项.

选项: 2.1.0 选项 GML 3 选项 ID 选项.



Note

ST_AsGML 选项 3 选项 TIN 选项.

- ✔ 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).

测试: 测试 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>
```

测试: 测试 3

```
-- Flip coordinates and output extended EPSG (16 | 1)--
SELECT ST_AsGML(3, ST_GeomFromText('POINT(5.234234233242 6.34534534534)',4326), 5, 17);
      st_asgml
      -----
      <gml:Point srsName="urn:ogc:def:crs:EPSG::4326"
><gml:pos
>6.34535 5.23423</gml:pos
></gml:Point>
```

```
-- Output the envelope (32) --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 32);
      st_asgml
      -----
      <gml:Envelope srsName="EPSG:4326">
        <gml:lowerCorner
>1 2</gml:lowerCorner>
        <gml:upperCorner
>10 20</gml:upperCorner>
      </gml:Envelope>
```

```
-- Output the envelope (32) , reverse (lat lon instead of lon lat) (16), long srs (1)= 32 | ←
16 | 1 = 49 --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 49);
      st_asgml
      -----
      <gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
        <gml:lowerCorner
>2 1</gml:lowerCorner>
        <gml:upperCorner
```

```
>20 10</gml:upperCorner>
</gml:Envelope>
```

```
-- Polyhedral Example --
SELECT ST_AsGML(3, ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0) ←
),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )''));
      st_asgml
-----
<gml:PolyhedralSurface>
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList>
        </gml:LinearRing>
      </gml:exterior>
    </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing>
          <gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
      <gml:PolygonPatch>
        <gml:exterior>
          <gml:LinearRing>
            <gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList>
            </gml:LinearRing>
          </gml:exterior>
        </gml:PolygonPatch>
        <gml:PolygonPatch>
          <gml:exterior>
            <gml:LinearRing>
              <gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList>
              </gml:LinearRing>
            </gml:exterior>
          </gml:PolygonPatch>
          <gml:PolygonPatch>
            <gml:exterior>
              <gml:LinearRing>
                <gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 0 0 1 0 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>
```


**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

例, 例, 例。

**Note**

返回的字符串格式为“X(经度) Y(纬度) 格式”。

经度范围 -180 到 180 度, 纬度范围 -90 到 90 度。

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。返回的字符串格式为 "SSS.SSSS" 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
```

返回的字符串格式为 "SSS.SSSS" 格式。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D°M'S.SSS"));
      st_aslatlon
-----
-2°19'29.928" -3°14'3.243"
```

返回的字符串格式为 "SSS.SSSS" 格式。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
      st_aslatlon
-----
2.3250 degrees S 3.2342 degrees W
```

返回的字符串格式为 "SSS.SSSS" 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.dddddd
- decimal degrees without cardinal direction: ddd.dddddd
- 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 简体中文.



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);
```

ST_AsMVT

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**.

Example

```
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;
```



```
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 — 将 TWKB(Tiny Well-Known Binary) 格式。

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);

描述

将 TWKB(Tiny Well-Known Binary) 格式。TWKB 是一种二进制格式，用于存储和传输几何数据。它比 WKB 更紧凑，并且支持更多的几何类型。

该函数接受一个几何对象（geom）并返回其 TWKB 表示。可选参数包括精度（prec）、精度 Z（prec_z）、精度 M（prec_m）、是否包含大小（with_sizes）以及是否包含边界框（with_boxes）。默认情况下，精度为 0，精度 Z 和 M 为 0，with_sizes 和 with_boxes 均为 false。

该函数还支持批量操作，接受一个几何数组（geom）和一个 ID 数组（ids）。批量操作允许一次性处理多个几何对象，并返回它们的 TWKB 表示。批量操作的精度和精度 Z/M 参数与单个操作相同。

该函数还支持批量操作，接受一个几何数组（geom）和一个 ID 数组（ids）。批量操作允许一次性处理多个几何对象，并返回它们的 TWKB 表示。批量操作的精度和精度 Z/M 参数与单个操作相同。



Note

<https://github.com/TWKB/Specification> 提供了 TWKB 的完整规范。有关 TWKB 的更多信息，请访问 <https://github.com/TWKB/twkb.js>。

Enhanced: 2.4.0 memory and speed improvements.

2.2.0 首次引入。



```
SELECT ST_AsTWKB('LINESTRING(1 1,5 5)::geometry');
          st astwkb
```

\x02000202020808

TWKB 1.0.0, "array_agg()" TWKB 1.0.0.

```
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 XXXXXX: ISO-IEC-19776-1.2-X3DEncodings-XML XXXXXX.

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' 000000000000. PostGIS 2.2 0000000, 000 X3D 0000000000000000
0, 0000 x/y 00000000000000000000. 00000 ST_AsX3D 000000000 (long,lat
or X,Y) 000000, X3D 0000000/000, y/x 0000000000.

- 0: X/Y (X,Y 非), (非).
- 1: X Y (GeoCoordinate) "latitude_first" (非).
- 2: WGS84 (SRID 4326) X3D GeoCoordinate geoSystem="GD" "WE" "longitude_first" X3D GeoCoordinate geoSystem="GD" "WE" "latitude_first", (2 + 1) = 3

PostGIS 几何类型	2D X3D 几何类型	3D X3D 几何类型
LINESTRING	二维多线串 PolyLine2D 几何类型。	LineSet
MULTILINESTRING	二维多线串 PolyLine2D 几何类型。	IndexedLineSet
MULTIPOINT	二维多点 Polypoint2D	PointSet
POINT	二维单点	IndexedPointSet
(MULTI) POLYGON, POLYHEDRALSURFACE	三维多面体 (markup) 几何类型。	IndexedFaceSet (面集) 几何类型。
TIN	TriangleSet2D (二维三角面集)	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 and <http://freewrl.sourceforge.net/> are available under the GNU GPL. FreeWRL_Launcher is available under the MIT license.

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.

Figure 2.2.0: A diagram showing a coordinate system with x and y axes. The x-axis is labeled 'x/y' and the y-axis is labeled 'y/x'. The origin is marked with a dot. The axes are labeled with 'x' and 'y' at the ends. The diagram is labeled 'Figure 2.2.0' in the top left corner.

- ✔ 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>

```

几何类型: 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>

```

几何类型: 多线串 (多线串集合)

```

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 — 返回给定 GeoHash 的 3D 表示。

Synopsis

text **ST_GeoHash**(geometry geom, integer maxchars=full_precision_of_point);

☐☐

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 ☐☐☐☐☐☐☐☐☐☐☐☐☐☐.



Note

ST_GeoHash requires input geometry to be in geographic (lon/lat) coordinates.



This method supports Circular Strings and Curves.

☐☐

```
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
```

☐☐

ST_GeomFromGeoHash, ST_PointFromGeoHash, ST_Box2dFromGeoHash

7.10 运算符 (operator)

7.10.1 Bounding Box Operators

7.10.1.1 &&

&& — A 2D 几何体 B 2D 几何体 TRUE 或 FALSE。

Synopsis

boolean **&&**(geometry A , geometry B);
boolean **&&**(geography A , geography B);

几何体

&& 几何体 A 2D 几何体 B 2D 几何体 TRUE 或 FALSE。



Note

几何体 (operand) 必须是 2D 几何体。

几何体: 2.0.0 版本 (polyhedral surface) 支持。

1.5.0 版本支持。

✔ This method supports Circular Strings and Curves.

✔ This function supports Polyhedral surfaces.

几何体

```
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)

几何体

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 INdices (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 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: `&&&`

```
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: `&&&`

```
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

`&&`

`&&`

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 );`

`&&`

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.

☒☒

```
SELECT ST_MakePoint(1,1,1) &&& ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) AS ↔
      overlaps;
```

```
overlaps
-----
t
(1 row)
```

☒☒

&&&(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);

☒☒

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.

例

```
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)
```

例

`&&(geometry,gidx), &&(gidx,geometry)`

7.10.1.9 &<

`&<` — A 与 B 相交是否返回 TRUE。

Synopsis

boolean `&<( geometry A , geometry B );`

例

`&<` 返回 A 与 B 相交是否返回 TRUE，如果 B 与 A 不相交，则返回 FALSE。



Note

如果 (operand) 与 B 相交，则返回 TRUE。

例

```
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)
```


Synopsis

boolean **&>**(geometry A , geometry B);

描述

&> 当 A 与 B 不相交, 返回 FALSE, 当 A 与 B 相交, 返回 TRUE。



Note

操作数 (operand) 必须是几何类型。

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, 返回 TRUE。

Synopsis

boolean **<<**(geometry A , geometry B);

描述

<< 当 A 与 B 不相交, 返回 TRUE。



Note

操作数 (operand) 必须是几何类型。

例

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 << tbl2.column2 AS left
FROM
  ( VALUES
    (1, 'LINESTRING (1 2, 1 5)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 3)::geometry),
    (3, 'LINESTRING (6 0, 6 5)::geometry),
    (4, 'LINESTRING (2 2, 5 6)::geometry)) AS tbl2;

column1 | column1 | left
-----+-----+-----
          1 |          2 | f
          1 |          3 | t
          1 |          4 | t
(3 rows)
```

例

>>, |>>, <<|

7.10.1.13 <<|

<<| — A 与 B 不相交返回 TRUE。

Synopsis

boolean <<|(geometry A , geometry B);

例

<<| 与 A 不相交 B 不相交返回 TRUE。



Note

(operand) 不相交。

例

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 <<| tbl2.column2 AS below
FROM
  ( VALUES
    (1, 'LINESTRING (0 0, 4 3)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (1 4, 1 7)::geometry),
    (3, 'LINESTRING (6 1, 6 5)::geometry),
    (4, 'LINESTRING (2 3, 5 6)::geometry)) AS tbl2;

column1 | column1 | below
-----+-----+-----
          1 |          2 | t
```

	1		3		f
	1		4		f
(3 rows)					

<<, >>, |>>

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

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 的几何对象。TRUE 表示包含。

Synopsis

boolean @(geometry A , geometry B);

例

@ 是否包含 B 的几何对象 A 的几何对象。TRUE 表示包含。



Note

几何对象 (operand) 必须是有效的几何对象。

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 INdexas (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)

例

```
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)

例

&&, &>, &<|, &<

7.10.1.21 |>>

|>> — A 是否严格在 B 上方。TRUE 表示是。

Synopsis

boolean |>>(geometry A , geometry B);

例

The |>> operator returns TRUE if the bounding box of geometry A is strictly above the bounding box of geometry B.



Note

如果 (operand) 是空值，则返回 NULL。

例

```
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.

例

```
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)
```

例

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [@&&\(geometry,box2df\)](#), [@&&\(box2df,geometry\)](#), [@&&\(box2df,box2df\)](#)

7.10.1.26 ~=

`~=` — A 与 B 是否相等 TRUE 或 FALSE。

Synopsis

boolean `~=`(geometry A , geometry B);

例

`~=` 返回 TRUE/FALSE A 与 B 是否相等 TRUE 或 FALSE。



Note

操作数 (operand) 必须是几何类型。

1.5.0 版本行为更改。



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](#).

图例

```
select 'LINESTRING(0 0, 1 1)::geometry ~= 'LINESTRING(0 1, 1 0)::geometry as equality;
equality      |
-----+-----
t            |
```

图例

ST_Equals, ST_OrderingEquals, =

7.10.2 图例 (operator)

7.10.2.1 <->

<-> — A 与 B 的 2 个最近邻点。

Synopsis

```
double precision <->( geometry A , geometry B );
double precision <->( geography A , geography B );
```

图例

<-> 返回 2 个最近邻点。 "ORDER BY" 子句 (index-assisted) 最近邻点 (nearest neighbor) 查询。 PostgreSQL 9.5 支持距离球 (distance sphere) 查询, 9.5 支持 KNN 查询 (distance sphere) 查询。 KNN 查询。



Note

操作数 (operand) 的 2 个 GiST 索引。 ORDER BY 子句 (index-assisted) 最近邻点 (nearest neighbor) 查询。



Note

图例, 图例 a.geom 图例 'SRID=3005;POINT(1011102 450541)::geometry' 图例, (图例/CTE(common table expression) 图例) 图例。

图例 **OpenGeo workshop: Nearest-Neighbour Searching** 图例。

图例: 2.2.0 图例 -- PostgreSQL 9.5 图例 KNN("K nearest neighbor") 图例。 图例 KNN 图例。 PostgreSQL 9.4 图例。

图例: 2.2.0 图例 -- PostgreSQL 9.5 图例 (Hybrid syntax) 图例。 PostgreSQL 2.2 图例, PostgreSQL 9.5 图例。

2.0.0 图例。 图例 KNN 图例。 PostgreSQL 9.1 图例。

图 10.1

```
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)

图 10.2 KNN 索引的查询结果:

```
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)

图 10.3 使用“EXPLAIN ANALYZE”查看 KNN 索引的查询计划。

PostgreSQL 9.5 引入了 KNN 索引，它使用 CTE(common table expression) 来优化查询，图 10.4 展示了 KNN 索引的查询计划。

```
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)		



ST DWithin, ST Distance, <#>

7.10.2.2 $|=$

$|=|$ — A $\boxtimes$ B $\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes$ (closest point of approach) $\boxtimes\boxtimes\boxtimes\boxtimes$ (trajectory) $\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes$.

Synopsis

```
double precision |( geometry A , geometry B );
```



$|=|$ `ST_IsValidTrajectory` (ST_IsValidTrajectory) 3 `ST_DistanceCPA` `ST_DistanceCPA` `ST_DistanceCPA`, `PostgreSQL 9.5.0` (PostgreSQL 9.5.0) N (nearest neighbor) `ST_DistanceCPA`.

Note

NOTE `(operand) N GiST ORDER BY` clause is supported by PostgreSQL 9.6+.

Note

`ST_GeomFromText('POINT(0 0)', 3005), ST_GeomFromText('LINESTRING(0 0 0,0 0 1)')::geometry`

2.2.0                                             PostgreSQL 9.5 (index-supported) .


```
-- 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!

Note

ORDER BY *expression* [*ASC* | *DESC*].



Note!

Note

```

g1.geom <#> 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, <->

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

(operand) N GiST ORDER BY .

Note

Note: `SELECT a.geom FROM a.geom SRID=3005;POINT(1011102 450541)::geometry`
`geom, (geom/CTE(common table expression) geom) geom`
`geom`.

2.2.0 实验环境. PostgreSQL 9.1 实现 KNN 实验.



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: $ST\_Contains(A, B) \Leftrightarrow (A \sqcap B = B) \wedge (Int(A) \sqcap Int(B) \neq \emptyset)$

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 简体中文

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.



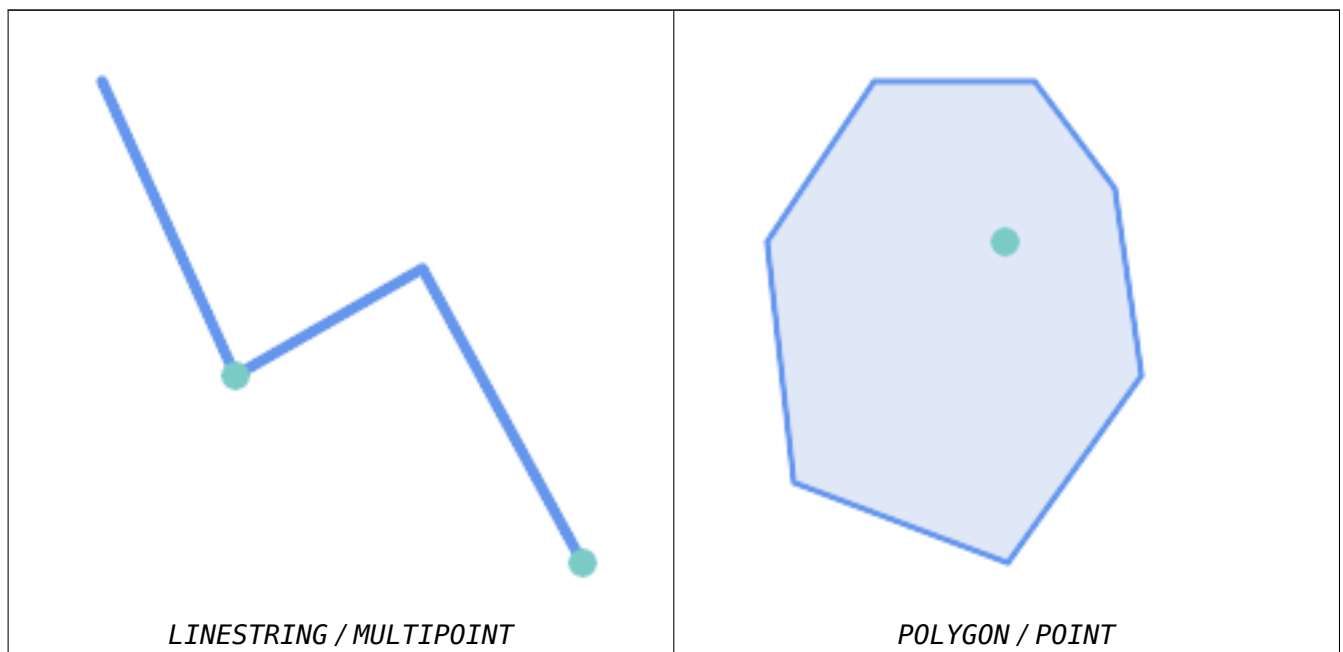
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3](#) - same as `within(geometry B, geometry A)`



This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

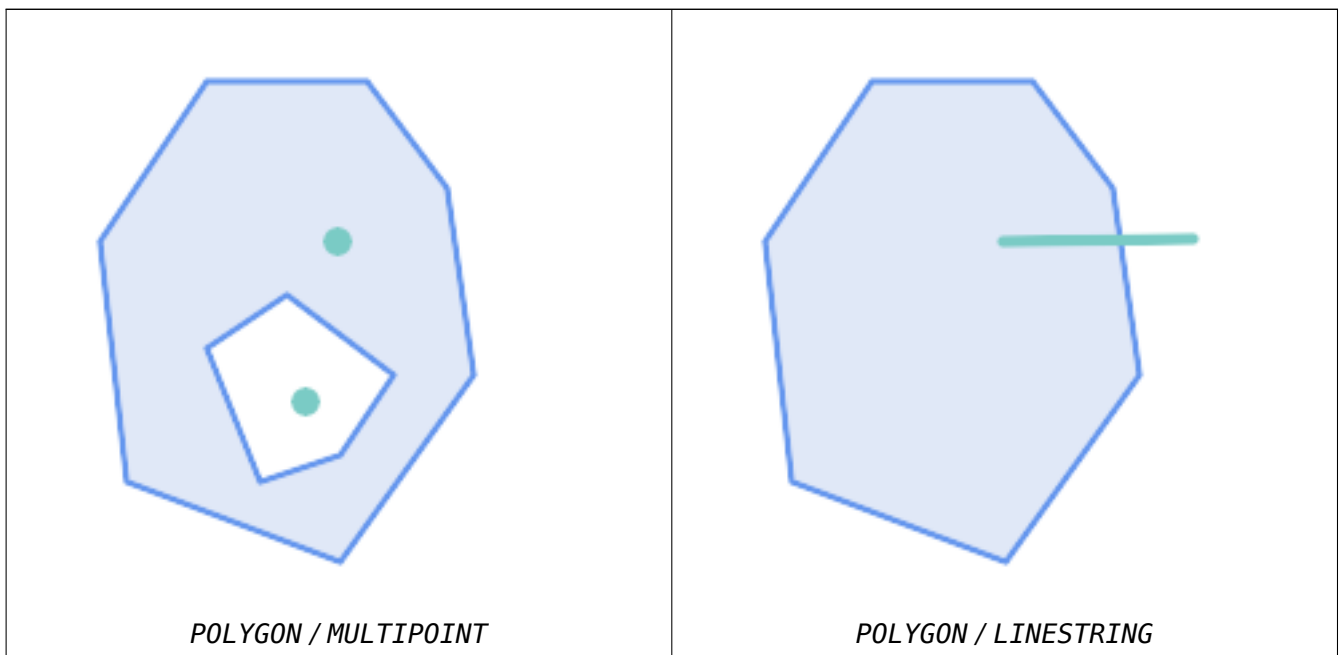
图例

`ST_Contains` returns TRUE in the following situations:

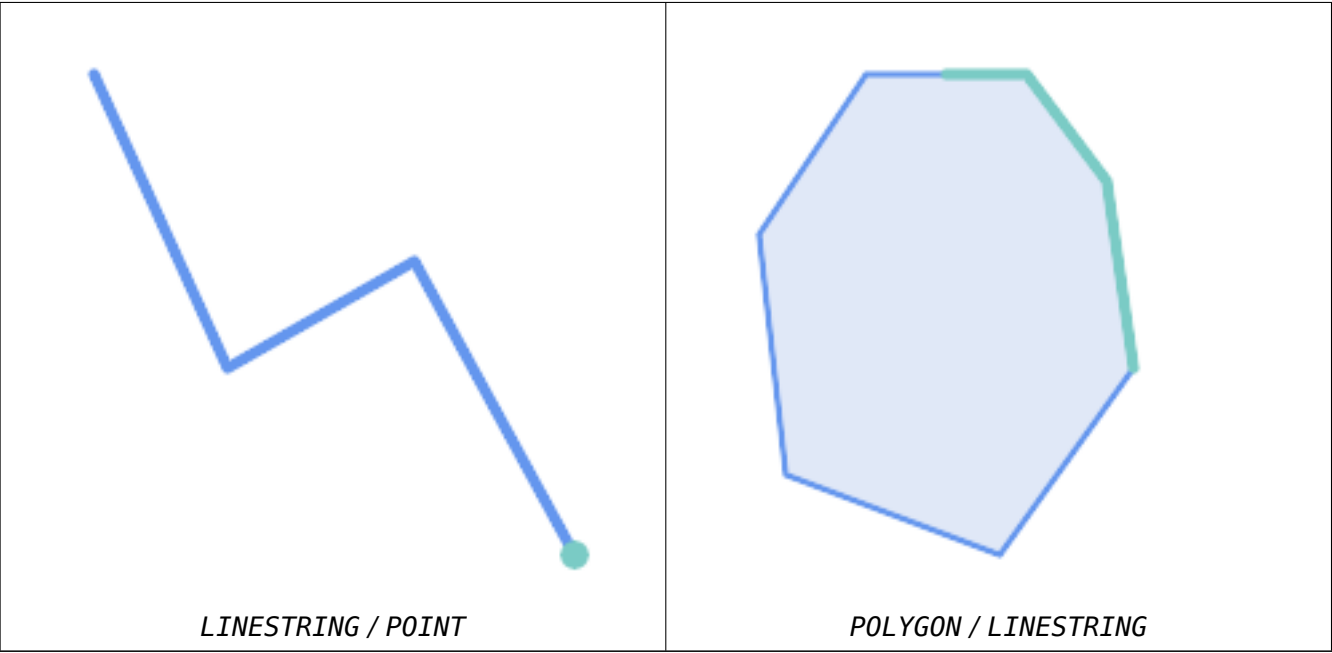




ST_Contains returns FALSE in the following situations:



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
```

☒☒

[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);

☒☒

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 3.10.0

1.4.0 简体中文.



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_GeometryType](#), [ST_Boundary](#), [ST_Contains](#), [ST_Covers](#), [ST_CoveredBy](#), [ST_Equals](#), [ST_Relate](#), [ST_Within](#)

7.11.1.4 ST_CoveredBy

ST_CoveredBy — Tests if every point of A lies in B

Synopsis

boolean **ST_CoveredBy**(geometry geomA, geometry geomB);
boolean **ST_CoveredBy**(geography geogA, geography geogB);

☒☒

Returns true if every point in Geometry/Geography A lies inside (i.e. intersects the interior or boundary of) Geometry/Geography B. Equivalently, tests that no point of A lies outside (in the exterior of) B.

In mathematical terms: $ST\_CoveredBy(A, B) \Leftrightarrow A \sqcap B = A$

`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



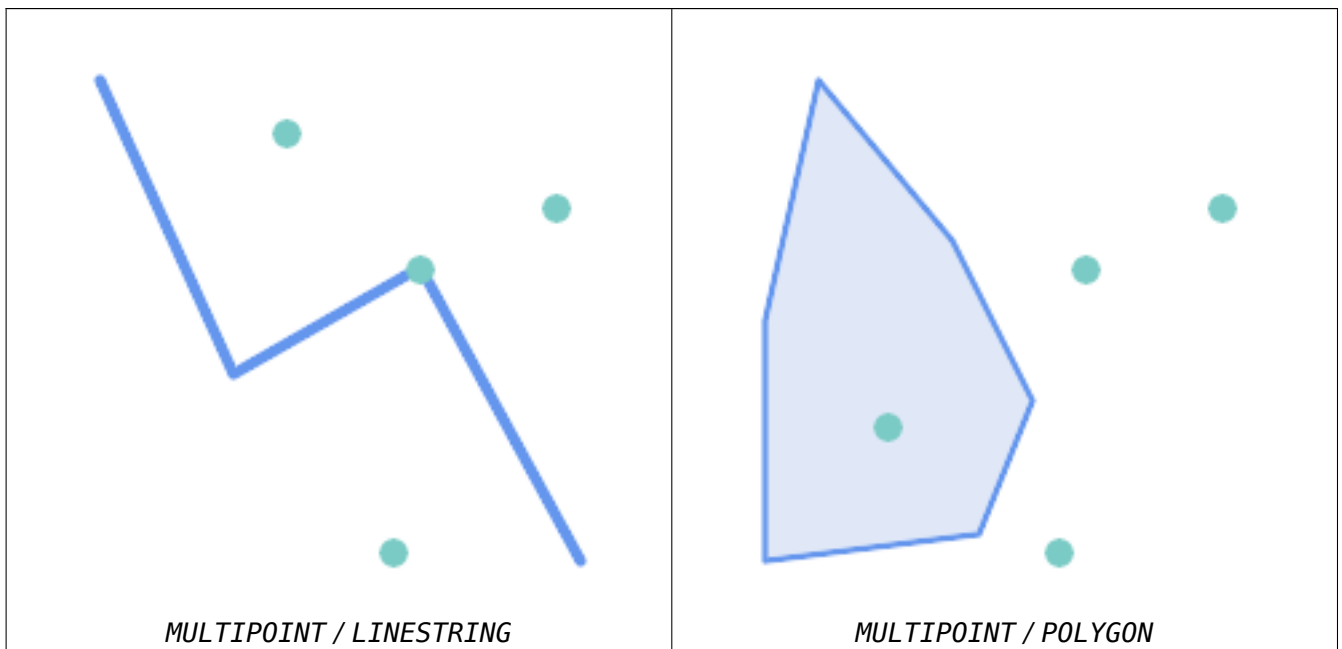
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.13.3](#)

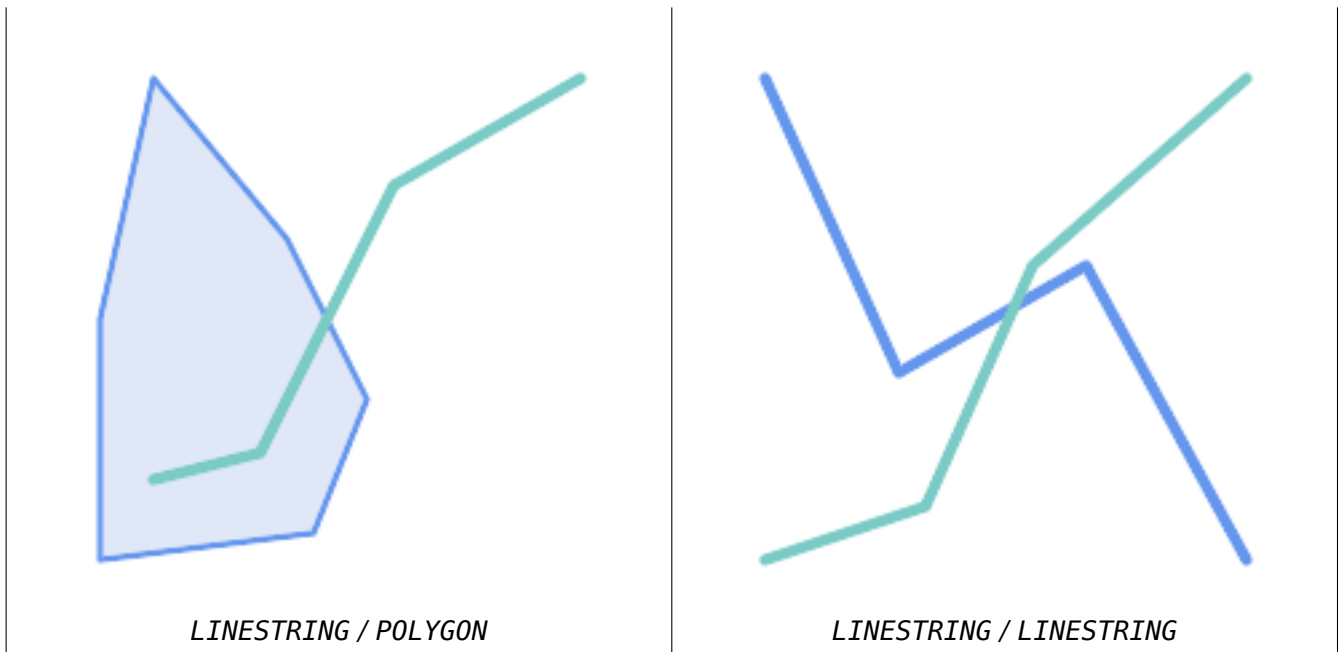


This method implements the SQL/MM specification. SQL-MM 3: 5.1.29



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).

图例

```
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)
```

图例

```
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
```

图例

&&, [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);

图例

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:

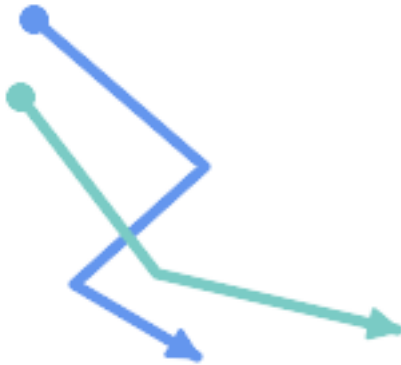
- 0: LINE NO CROSS
- -1: LINE CROSS LEFT

- 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

国国

Example: LINE CROSS LEFT and LINE CROSS RIGHT

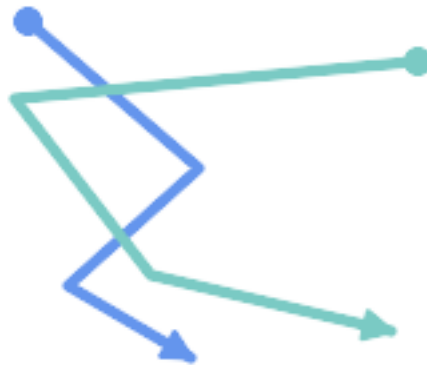


Blue: Line A; Green: Line B

```
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

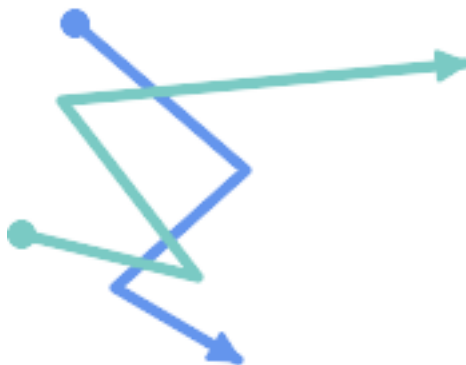


Blue: Line A; Green: Line B

```
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/arcsgis/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

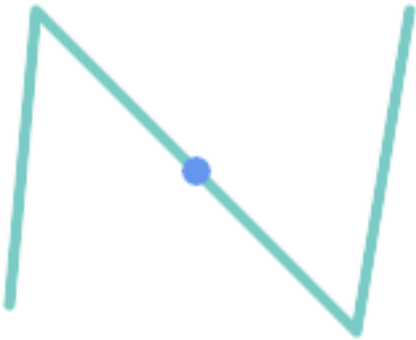
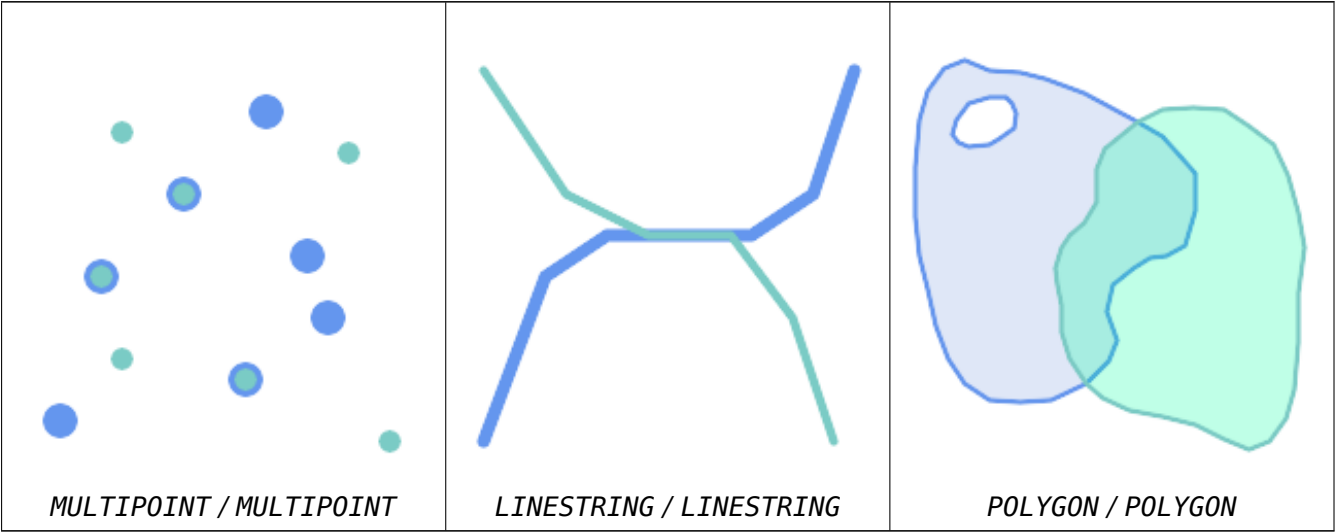
Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

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 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

SQL

```
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

SQL

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);
```

SQL

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`



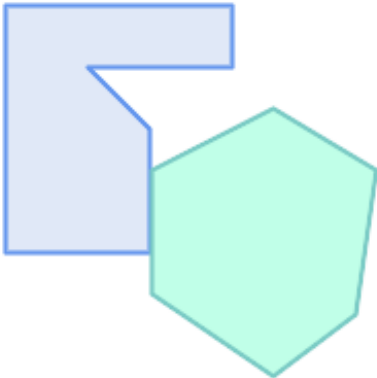
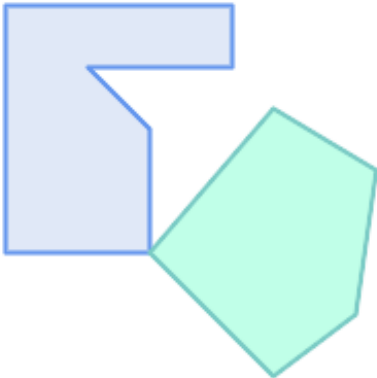
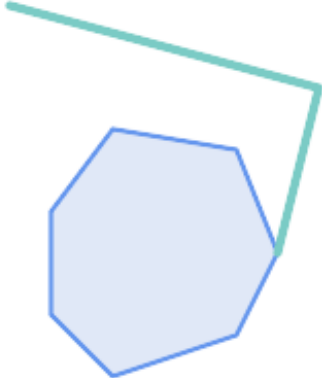
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3](#)

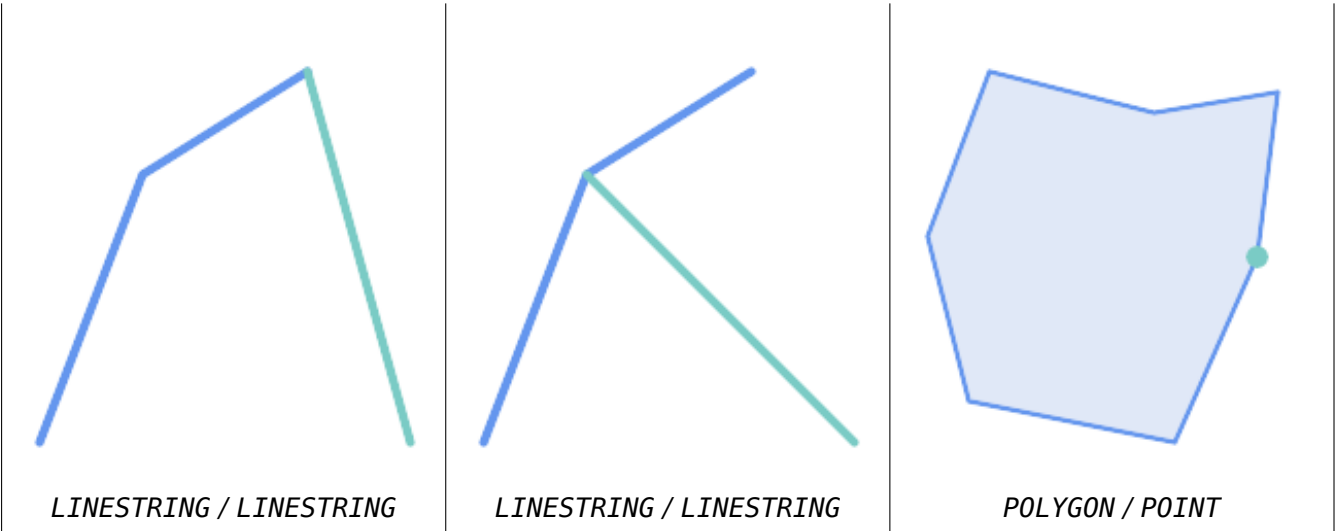


This method implements the SQL/MM specification. SQL-MM 3: 5.1.28



The `ST_Touches` predicate returns TRUE in the following examples.

 <i>POLYGON / POLYGON</i>	 <i>POLYGON / POLYGON</i>	 <i>POLYGON / LINESTRING</i>



```
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: $ST\_Within(A, B) \Leftrightarrow (A \sqcap B = A) \wedge (Int(A) \sqcap Int(B) \neq \emptyset)$

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 简体中文

☒☒

```
-- 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


```
SELECT ST_PointInsideCircle(ST_Point(1,2), 0.5, 2, 3);
 st_pointinsidecircle
-----
t
```


ST DWithin

7.12 Measurement Functions

7.12.1 ST_Area

ST Area — □□□□□□□□□□□□□□.

Synopsis

```
float ST_Area(geometry g1);
float ST_Area(geography geog, boolean use_spheroid = true);
```


ST_Surface(*geom*) - ST_MultiSurface(*geom*) - *geom*. *geom*, SRID *SRID* 2 *geom*. *geom*, *geom* (curved surface) *geom*. *geom*, ST_Area(*geom*,false) *geom*.

2.0.0 2 (polyhedral surface)

GeographicLib 2.2.0, Proj 4.9.0.

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

XXXXXXXXXX, (2.5 XXXXX) 2 XXXXXXXXXXXXXXXX. 2.5 XXXXX 0 XXX (non-zero) XXXXXXXXXXXXXXXX, XY XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.



 (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。圖中所有之點位均係根據該座標系統計算所得。圖中所有之點位均係根據該座標系統計算所得。圖中所有之點位均係根據該座標系統計算所得。

```
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 函数使用 3 个参数 (最长) 来指定要移动的距离。
。 [Plpgsqlfunctions PostGIS wiki section](#) 的 upgis_lineshift 函数。

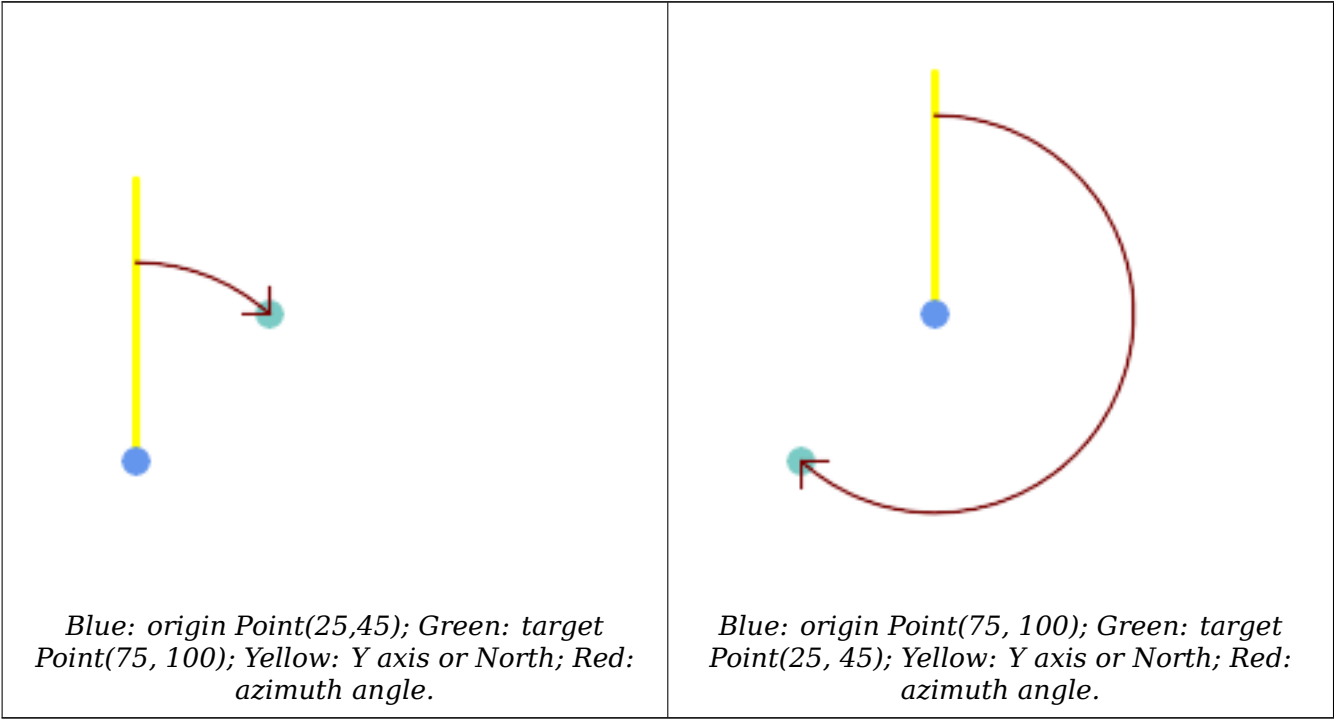
- 1.1.0 版本。
- 2.0.0 版本。
- 2.2.0 版本使用 GeographicLib 库。 使用 Proj 4.9.0 版本。

。

。

```
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 — 返回 3 个参数 (最长) 的方位角。

Synopsis

float **ST_Angle**(geometry point1, geometry point2, geometry point3, geometry point4);
float **ST_Angle**(geometry line1, geometry line2);

返回

返回角度 3 点 (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。返回角度 3 点 (longest) 的度数。

Note that `ST_Angle(P1,P2,P3) = ST_Angle(P2,P1,P2,P3)`.

Availability: 2.5.0

返回

返回角度 3 点 (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);

返回

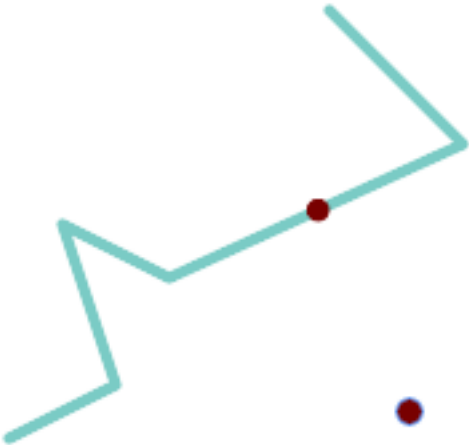
Returns the 2-dimensional point on geom1 that is closest to geom2. This is the first point of the shortest line between the geometries (as computed by **ST_ShortestLine**).



Note
3 返回 **ST_3DClosestPoint** 返回。

Enhanced: 3.4.0 - Support for geography.
1.5.0 返回。

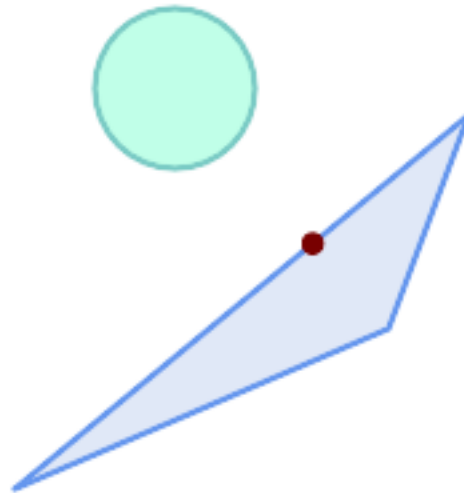
返回



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 ☒☒☒☒☒☒☒☒☒☒☒.

OpenGL: 2.2.0 OpenGL 2D Renderer, (OpenGL Z 0 Renderer) 2D
 OpenGL. 2D 3D, OpenGL Z 0 Renderer.



7.12.5 ST_ClosestPoint -- 3D, 2D 最近点 (closest point)

```
SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,
       ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt
FROM (SELECT 'POINT(100 100 30)::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)::'::
       geometry As line
      ) As foo;
```

cp3d_line_pt		
cp2d_line_pt		
-----+-----		

POINT(54.6993798867619 128.935022917228 11.5475869506606)		POINT(73.0769230769231 115.384615384615)

7.12.5 ST_ClosestPoint -- 3D, 2D 最近点 (closest point)

```
SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,
       ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)::'::
       geometry As line
      ) As foo;
```

cp3d_line_pt		cp2d_line_pt
-----+-----		

POINT(54.6993798867619 128.935022917228 11.5475869506606)		POINT(50 75)

7.12.5 ST_ClosestPoint -- 3D, 2D 最近点 (closest point)

```
SELECT ST_AsEWKT(ST_3DClosestPoint(poly, mline)) As cp3d,
       ST_AsEWKT(ST_ClosestPoint(poly, mline)) As cp2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,
100 100 5, 175 150 5))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125
100 1, 175 155 1),
       (1 10 2, 5 20 1))') As mline ) As foo;
       cp3d | cp2d
-----+-----
POINT(39.993580415989 54.1889925532825 5) | POINT(20 40)
```

7.12.6

ST_AsEWKT, ST_ClosestPoint, ST_3DDistance, ST_3DShortestLine

7.12.6 ST_Distance

ST_Distance — 3D 最长距离 (longest) 最近点。

Synopsis

```
float ST_Distance(geometry g1, geometry g2);
float ST_Distance(geography geog1, geography geog2, boolean use_spheroid = true);
```

几何

几何类型, 几何类型 3 维几何类型 (SRS ID) 几何类型.

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 几何类型. 几何类型. 几何类型. 几何类型. 几何类型. 几何类型. 几何类型. 几何类型. 几何类型. 几何类型.

Enhanced: 2.1.0 improved speed for geography. See [Making Geography faster](#) for details.

几何: 2.1.0 几何类型.

几何: 2.2.0 几何类型. GeographicLib 几何类型. 几何类型. Proj 4.9.0 几何类型.

Changed: 3.0.0 - does not depend on SFCGAL anymore.

几何

Geometry example - units in planar degrees 4326 is WGS 84 long lat, units are degrees.

```
SELECT ST_Distance(
  'SRID=4326;POINT(-72.1235 42.3521)::geometry',
  'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry ');
-----
0.00150567726382282
```

Geometry example - units in meters (SRID: 3857, proportional to pixels on popular web maps). Although the value is off, nearby ones can be compared correctly, which makes it a good choice for algorithms like KNN or KMeans.

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry', 3857),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry', 3857) ) ↵
;
-----
167.441410065196
```

Geometry example - units in meters (SRID: 3857 as above, but corrected by cos(lat) to account for distortion)

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry', 3857),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry', 3857)
) * cosd(42.3521);
-----
123.742351254151
```

Geometry example - units in meters (SRID: 26986 Massachusetts state plane meters) (most accurate for Massachusetts)

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry, 26986),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry, 26986) ) ←
  );
-----
123.797937878454
```

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (least accurate)

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry, 2163),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry, 2163) ) ←
  ;
-----
126.664256056812
```

地理

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
```

地理

[ST_3DDistance](#), [ST_DWithin](#), [ST_DistanceSphere](#), [ST_DistanceSpheroid](#), [ST_MaxDistance](#), [ST_HausdorffDistance](#), [ST_FrechetDistance](#), [ST_Transform](#)

7.12.7 ST_3DDistance

ST_3DDistance — 计算两个 3D 几何体 (SRS 为 3D) 之间的 3D 距离。

Synopsis

float **ST_3DDistance**(geometry g1, geometry g2);

地理

计算两个 3D 几何体 (SRS 为 3D) 之间的 3D 距离。



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 文档.
- 文档: 2.2.0 文档, 2D 与 3D 文档 Z 文档 Z 与 0 文档.
- Changed: 3.0.0 - SFCGAL version removed

文档

```
-- 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
```

文档

[ST_Distance](#), [ST_3DClosestPoint](#), [ST_3DDWithin](#), [ST_3DMaxDistance](#), [ST_3DShortestLine](#), [ST_Transform](#)

7.12.8 ST_DistanceSphere

`ST_DistanceSphere` — 文档. PostGIS 1.5 文档.

Synopsis

float **ST_DistanceSphere**(geometry geom1lonlatA, geometry geom1lonlatB, float8 radius=6371008);

图

图 2 展示了使用 `ST_DistanceSpheroid` 函数。SRID 指定了使用的椭球体。PostGIS 1.5 之前，该函数使用 `ST_DistanceSphere` 函数。

1.5 版本之前，该函数使用 `ST_DistanceSphere` 函数。1.5 版本之后，该函数使用 `ST_DistanceSpheroid` 函数。

图 2.2.0 展示了 `ST_Distance_Sphere` 函数。

图

```
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
```

图

`ST_Distance`, `ST_DistanceSpheroid`

7.12.9 ST_DistanceSpheroid

`ST_DistanceSpheroid` — 计算两个几何体之间的球面距离。PostGIS 1.5 之前，该函数使用 `ST_DistanceSphere` 函数。

Synopsis

float **ST_DistanceSpheroid**(geometry geom1onlatA, geometry geom1onlatB, spheroid measurement_spheroid)

图

图 展示了使用 `ST_DistanceSpheroid` 函数。该函数使用 `ST_LengthSpheroid` 函数。PostGIS 1.5 之前，该函数使用 `ST_LengthSphere` 函数。



Note

图 展示了 SRID 指定了使用的椭球体。图 展示了使用 `ST_DistanceSpheroid` 函数。

1.5 使用 `ST_DistanceSphere` 函数。1.5 使用 `ST_DistanceSphere` 函数。

2.2.0 使用 `ST_Distance_Spheroid` 函数。

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) 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` 函数。使用 `ST_FrechetDistance` 函数。

**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 3.7.0

Availability: 2.4.0 - requires GEOS >= 3.7.0

SQL

```
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)
```

SQL

ST_HausdorffDistance

7.12.11 ST_HausdorffDistance

`ST_HausdorffDistance` — 返回两个几何体的 3 维 Hausdorff (shortest) 距离。

Synopsis

```
float ST_HausdorffDistance(geometry g1, geometry g2);
float ST_HausdorffDistance(geometry g1, geometry g2, float densifyFrac);
```

SQL

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);
```



ST FrechetDistance

7.12.12 ST_Length

ST_Length — .

Synopsis

```
float ST_Length(geometry a_2dlinestring);
float ST_Length(geography geog, boolean use_spheroid = true);
```


注意: 曲线, 多曲线 ST_Curve, ST_MultiCurve 返回 2 个值。
 第一个值 0 表示成功。第二个值 **ST_Perimeter** 表示周长。

For geography types: computation is performed using the inverse geodesic calculation. Units of length are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If `use_spheroid = false`, then the calculation is based on a sphere instead of a spheroid.

`ST_Length2D`, `ST_Length2D_Spheroid`.



Warning

0000: 2.0.0 0000000000000000. 2.0.0 0000000000000000/000000
 0000000000000000/0000000000000000. 2.0.0 00000000000000
 0 000000000000. 0000000000 ST Perimeter 00000000.



Note

ST Length(gg,false);



✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
s2.1.5.1



✔ This method implements the SQL/MM specification. SQL-MM 3: 7.1.2, 9.3.4 1.5.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒.



XXXXXXXXXXXXXXXXXXXXX. XXX EPSG:2249 XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX.

```
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
```

图例

WGS84 图例.

```
-- 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
```

图例

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 的平方。

Synopsis

float **ST_Length2D**(geometry a_2dlinestring);

图例

图例 2D 平面上的长度。返回 ST_Length 的平方。

ST_Length, ST_3DLength

ST_3DLength — .

```
float ST_3DLength(geometry a_3dlinestring);
```

[illegible]

- ✔ 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
- 📦📦📦📦: 2.0.0 📦📦📦📦📦📦 ST_Length3D 📦📦📦📦📦📦📦📦.

[illegible]

```
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
```

ST_Length, ST_Length2D

ST LengthSpheroid — .

```
float ST_LengthSpheroid(geometry a geometry, spheroid a spheroid);
```


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

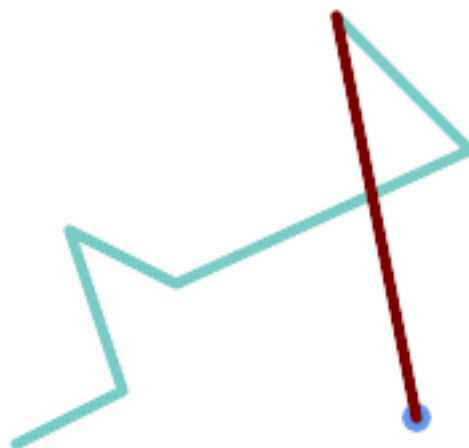
Returns the 2-dimensional longest line between the points of two geometries. The line returned starts on g1 and ends on g2.

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](#).

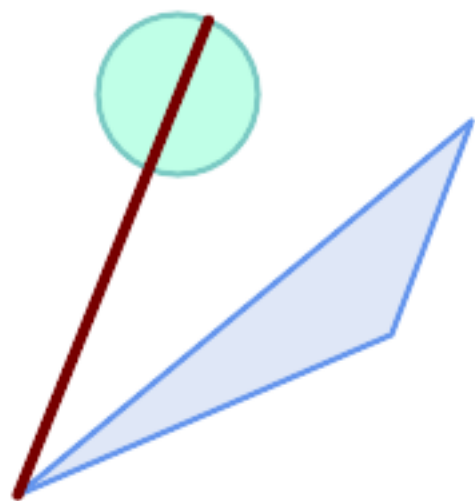
1.5.0 引入。

返回



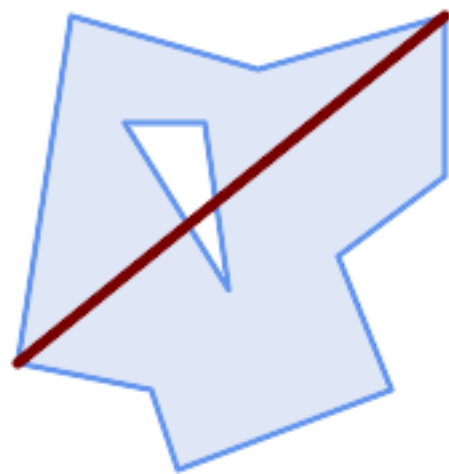
返回

```
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)
```



最長線與最小外接圓

```
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
```


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)
```

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);
```


(ST_IsValid) ST_IsValid (ST_IsSimple) ST_IsSimple, (WKT, KML, GML GeoJSON) (MapInfo TAB)

The minimum clearance is a quantitative measure of a geometry’s robustness to change in coordinate precision. It is the largest distance by which vertices of the geometry can be moved without creating an invalid geometry. Larger values of minimum clearance indicate greater robustness.

☐ e ☐ , ☐

- [illegible]

ST Minimum Clearance 0.75m.

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
```

返回

[ST_MinimumClearanceLine](#), [ST_Crosses](#), [ST_Dimension](#), [ST_Intersects](#)

7.12.21 ST_MinimumClearanceLine

`ST_MinimumClearanceLine` — 返回 2 个点的 LineString，表示几何体的最小 clearance。

Synopsis

Geometry **ST_MinimumClearanceLine**(geometry g);

返回

Returns the two-point LineString spanning a geometry's minimum clearance. If the geometry does not have a minimum clearance, LINESTRING EMPTY is returned.

GEOS 版本

2.3.0 版本。GEOS 3.6.0 版本。

返回

```
SELECT ST_AsText(ST_MinimumClearanceLine('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))'));
-----
LINESTRING(0.5 0.00032,0.5 0)
```

返回

[ST_MinimumClearance](#)

7.12.22 ST_Perimeter

`ST_Perimeter` — Returns the length of the boundary of a polygonal geometry or geography.

Synopsis

float **ST_Perimeter**(geometry g1);
float **ST_Perimeter**(geography geog, boolean use_spheroid = true);

返回

对于 ST_Surface, ST_MultiSurface(几何体, 几何体) 返回 2 个点的 LineString。返回 0 个点的 LineString。返回 [ST_Length](#) 返回的几何体。返回的几何体，返回的几何体。

For geography types, the calculations are performed using the inverse geodesic problem, where perimeter units are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is

specified by the SRID, otherwise it is exclusive to WGS84. If `use_spheroid = false`, then calculations will approximate a sphere instead of a spheroid.

`ST_Perimeter2D` 返回多边形、多线、多点的周长，使用平面几何。

✔ 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 返回类型: float。

用法:

计算一个四边形的周长。使用 EPSG:2249 平面坐标系。

```
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:

计算一个四边形的周长。使用 WGS84 球面坐标系。

```
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);
```

[illegible]

Note

```

XXXXXXXXXX ST_Perimeter XXXXXXXXXX. XXXXXXXXXX ST_Perimeter XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXXXXXXX.

```



ST Perimeter

7.12.24 ST 3DPerimeter

ST 3DPerimeter — □□□□□□□□□□□□□□.

Synopsis

```
float ST_3DPerimeter(geometry geomA);
```

函数

返回几何体的 3D 周长。2D 周长使用 `ST_Perimeter2D` 函数。



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: `double`。

函数

返回几何体的 3D 周长、2D 周长和 2D 周长。

```
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

函数

`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 2D 线。

Synopsis

geometry **ST_ShortestLine**(geometry geom1, geometry geom2);
 geography **ST_ShortestLine**(geography geom1, geography geom2, boolean use_spheroid = true);

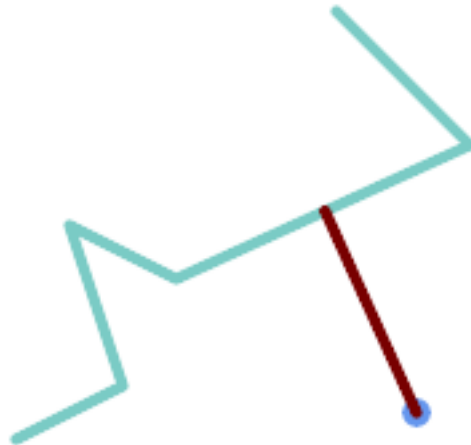
函数

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.

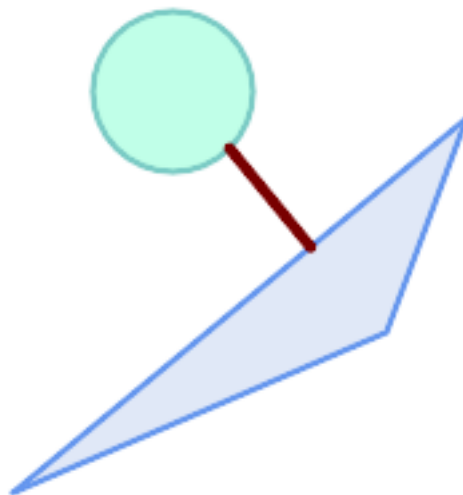
1.5.0 函数名: `ST_ShortestLine`。

圖



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)
```

返回

`ST_ClosestPoint`, `ST_Distance`, `ST_LongestLine`, `ST_MaxDistance`

7.12.26 ST_3DShortestLine

`ST_3DShortestLine` — 返回 3 维最短 (shortest) 连接线的几何体。

Synopsis

geometry **ST_3DShortestLine**(geometry g1, geometry g2);

返回

返回 3 维最短 (shortest) 连接线的几何体。该函数返回最短连接线的几何体，该几何体是 3 维的。g1 和 g2 可以是任何 3 维几何体，包括点、线、面。g1 和 g2 可以是任何 3 维几何体，包括点、线、面。该函数返回最短连接线的几何体，该几何体是 3 维的。g1 和 g2 可以是任何 3 维几何体，包括点、线、面。该函数返回最短连接线的几何体，该几何体是 3 维的。g1 和 g2 可以是任何 3 维几何体，包括点、线、面。

2.0.0 版本引入。

注意: 2.2.0 版本引入 2D 最短连接线的函数，(返回最短连接线的 Z 为 0 的 2D 几何体) 2D 最短连接线的函数。2D 和 3D 最短连接线的函数，返回 Z 为 0 的 2D 几何体。

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

✓ This function supports Polyhedral surfaces.

返回

返回 3D, 2D 最短连接线的几何体

```
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)::' AS
       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)



```
-- 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](#)

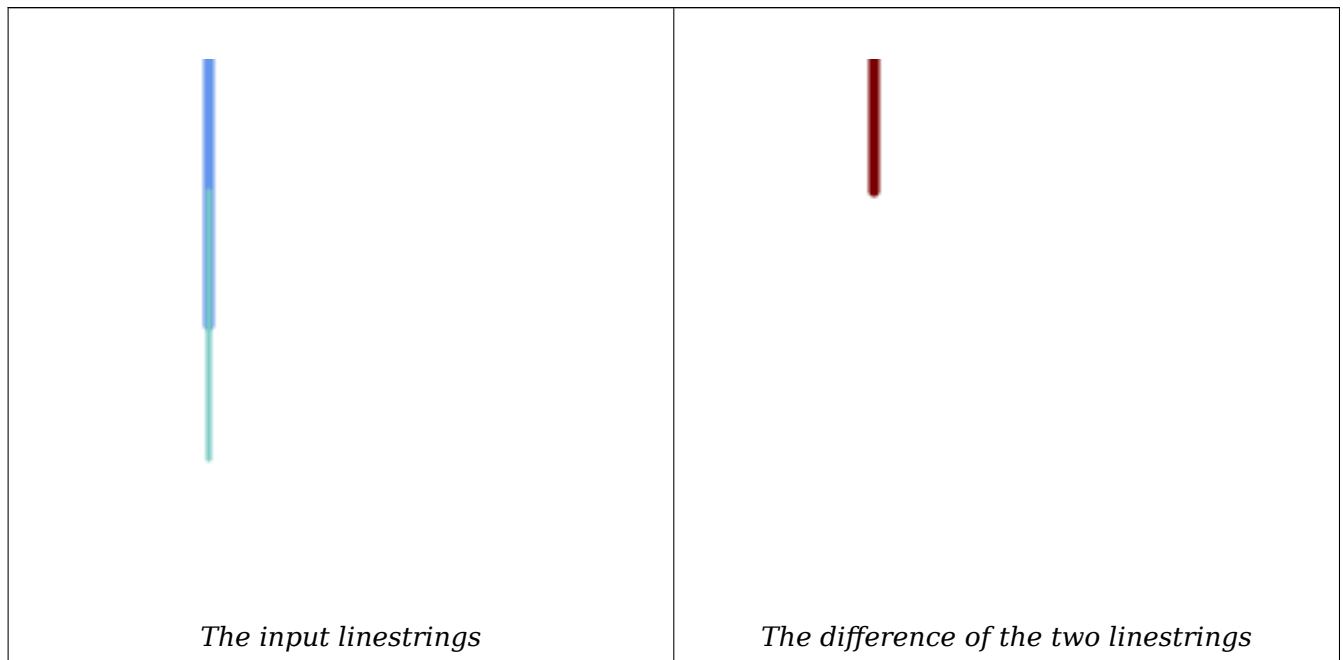


This method implements the SQL/MM specification. SQL-MM 3: 5.1.20



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.

国国



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)
```

国国

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 );
```



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 [2.4.0](#)

2.0.0 [2.4.0](#).

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.

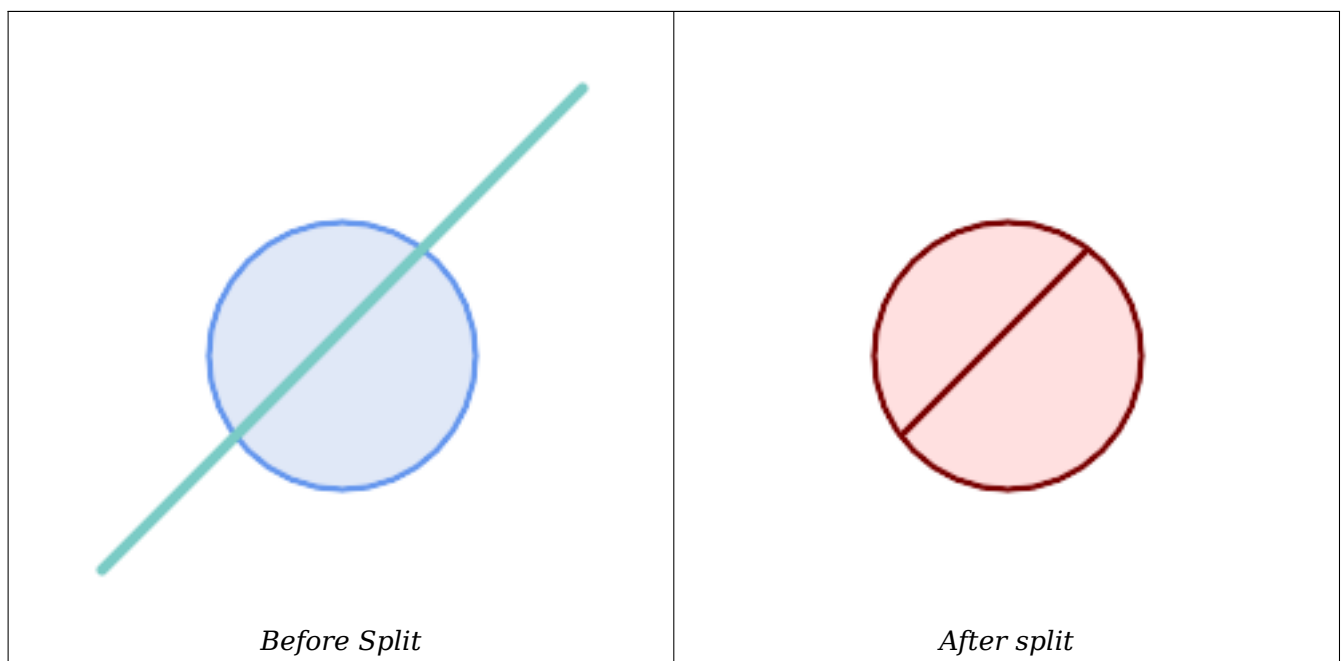
Availability: 2.0.0 requires GEOS

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.

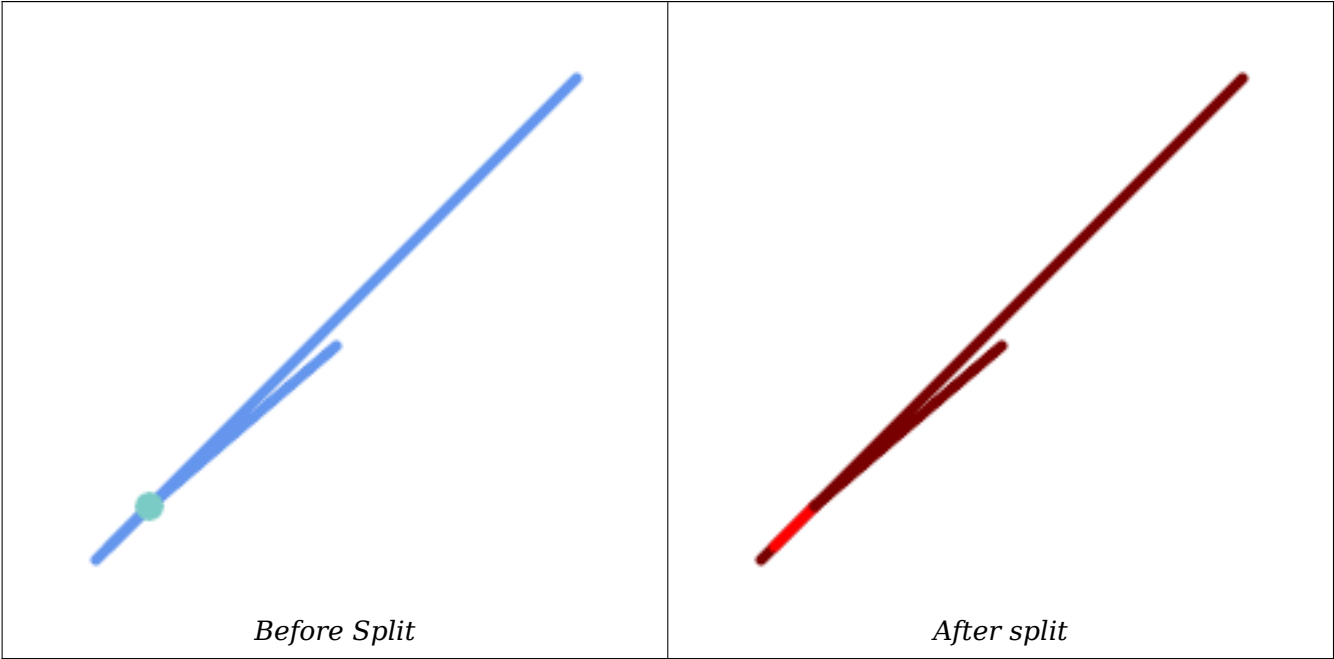


```
SELECT ST_AsText( ST_Split(
    ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50), -- circle
    ST_MakeLine(ST_Point(10, 10),ST_Point(190, 190)) -- line
));

-- 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(
  LINESTRING(0 0,100 100))
```

[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   .

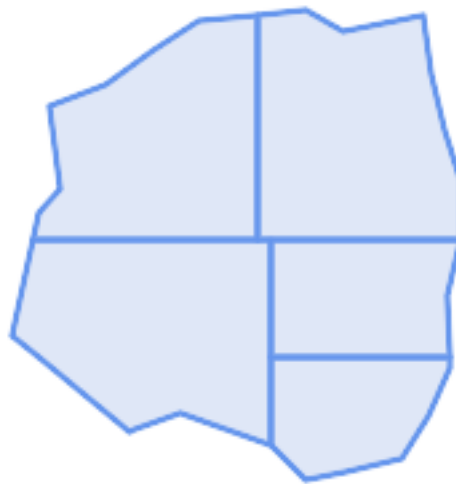
Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5.

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.

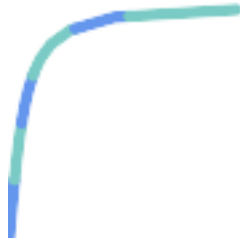


Subdivided to maximum 10 vertices

```
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.

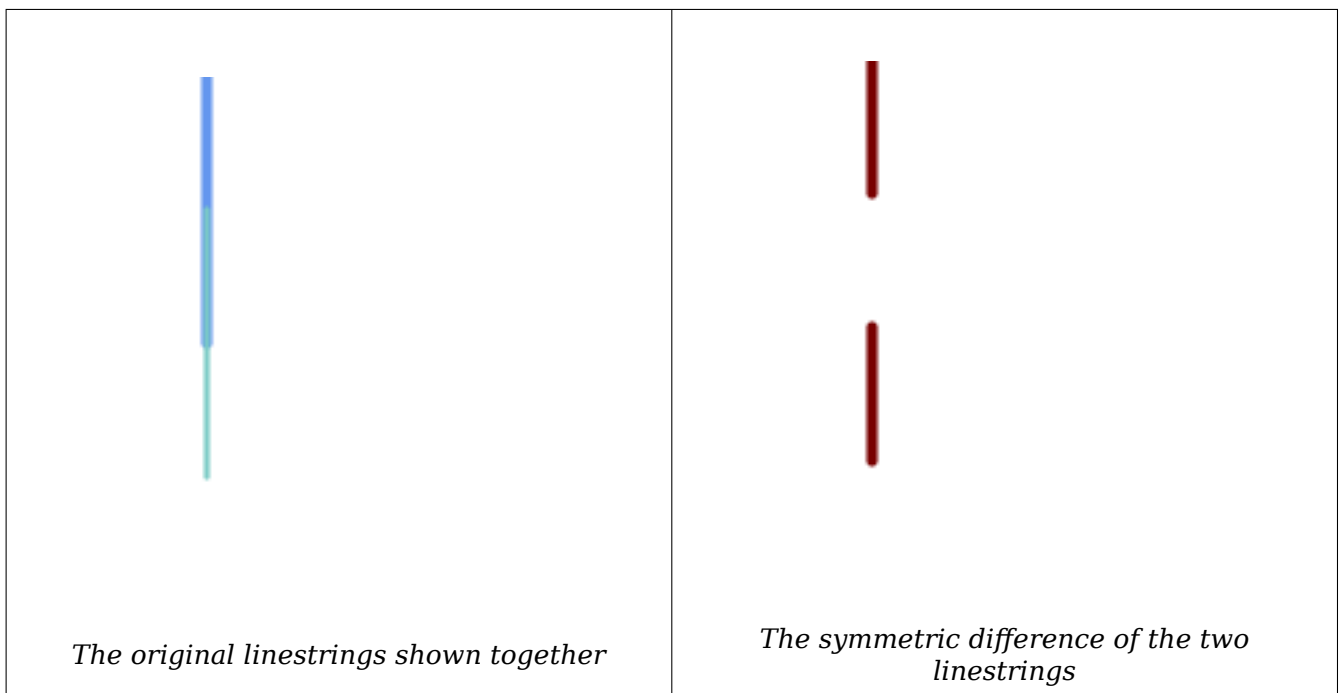
Requires GEOS $\geq$ 3.9.0 to use the `gridSize` parameter

✔ 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))
```

囧囧

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 几何操作

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 output is always a valid polygonal geometry. Buffer can handle invalid inputs, so buffering by distance 0 is sometimes used as a way of repairing invalid polygons. [ST_MakeValid](#) can also be used for this purpose.

**Note**

Buffering is sometimes used to perform a within-distance search. For this use case it is more efficient to use [ST_DWithin](#).

**Note**

This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry.

Enhanced: 2.5.0 - `ST_Buffer` geometry support was enhanced to allow for side buffering specification `side=both|left|right`.

Availability: 1.5 - `ST_Buffer` was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added.

GEOS 

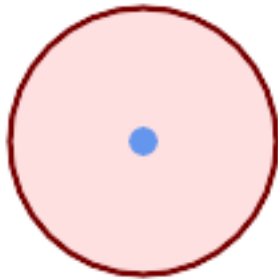


This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3](#)



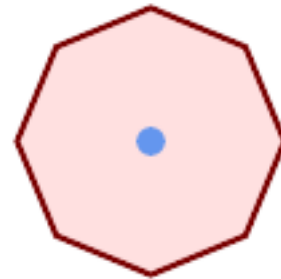
This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.30





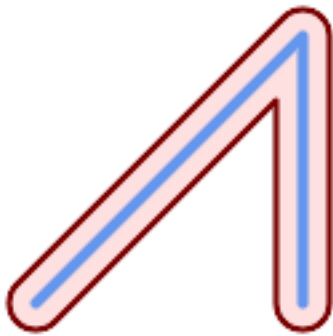
quad_segs=8 (8段)

```
SELECT ST_Buffer(  
  ST_GeomFromText('POINT(100 90)'),  
  50, 'quad_segs=8');
```



quad_segs=2 (2段)

```
SELECT ST_Buffer(  
  ST_GeomFromText('POINT(100 90)'),  
  50, 'quad_segs=2');
```



endcap=round join=round (8段)

```
SELECT ST_Buffer(  
  ST_GeomFromText(  
    'LINESTRING(50 50,150 150,150 50)'  
  ), 10, 'endcap=round join=round');
```



endcap=square

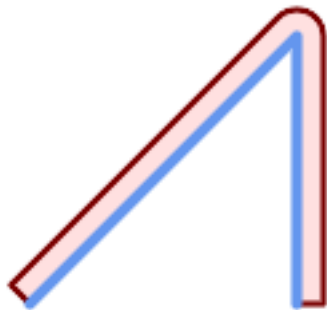
```
SELECT ST_Buffer(  
  ST_GeomFromText(  
    'LINESTRING(50 50,150 150,150 50)'  
  ), 10, 'endcap=square join=round');
```

*join=bevel*

```
SELECT ST_Buffer(  
  ST_GeomFromText(  
    'LINESTRING(50 50,150 150,150 50)'  
  ), 10, 'join=bevel');
```

*join=mitre mitre_limit=5.0 (简体中文)*

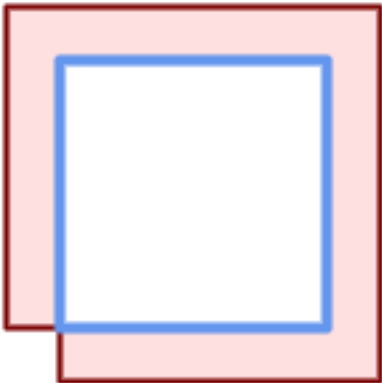
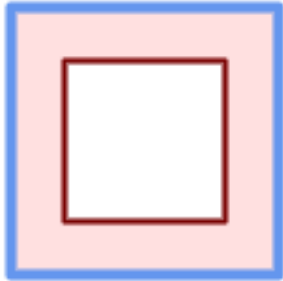
```
SELECT ST_Buffer(  
  ST_GeomFromText(  
    'LINESTRING(50 50,150 150,150 50)'  
  ), 10, 'join=mitre mitre_limit=5.0');
```

*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');
```

 <i>right-hand-winding, polygon boundary side=left</i> <pre>SELECT ST_Buffer(ST_ForceRHR(ST_Boundary(ST_GeomFromText('POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))')'),), 20, 'side=left');</pre>	 <i>right-hand-winding, polygon boundary side=right</i> <pre>SELECT ST_Buffer(ST_ForceRHR(ST_Boundary(ST_GeomFromText('POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))')'),), 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](#).

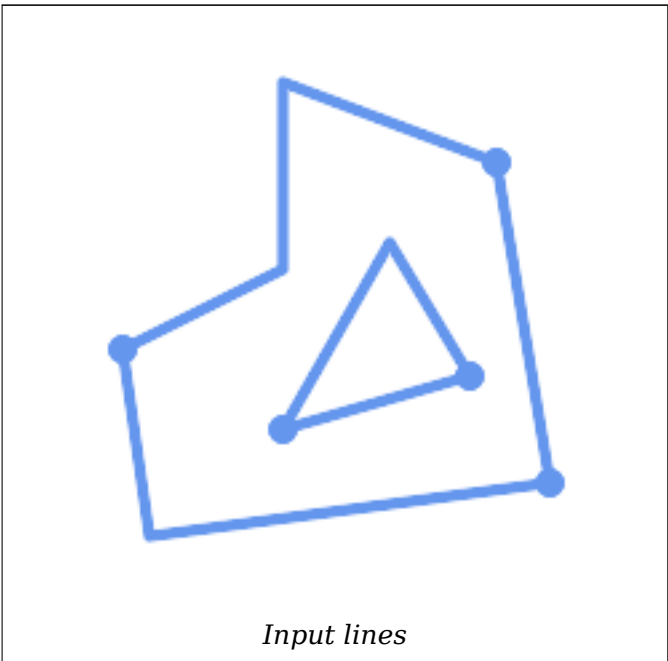


Note

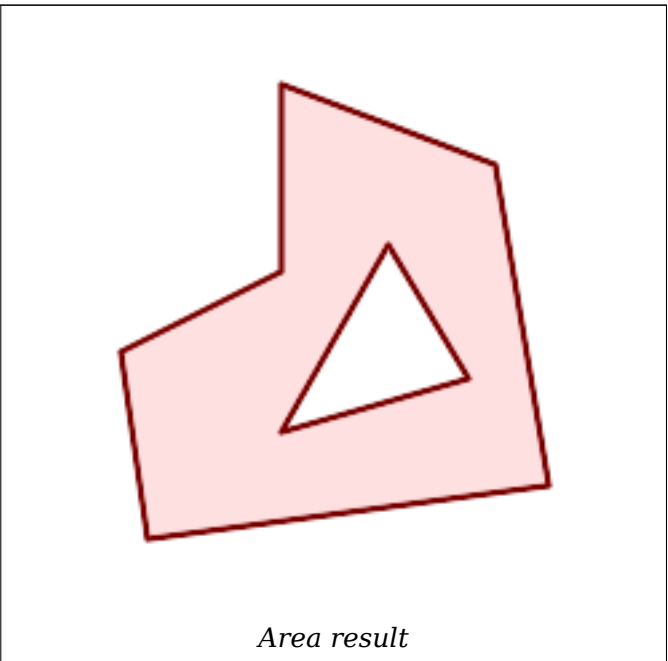
Input linework must be correctly noded for this function to work properly. [ST_Node](#) can be used to node lines.

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



Input lines



Area result

```

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 — 返回几何图形的质心。

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](#).

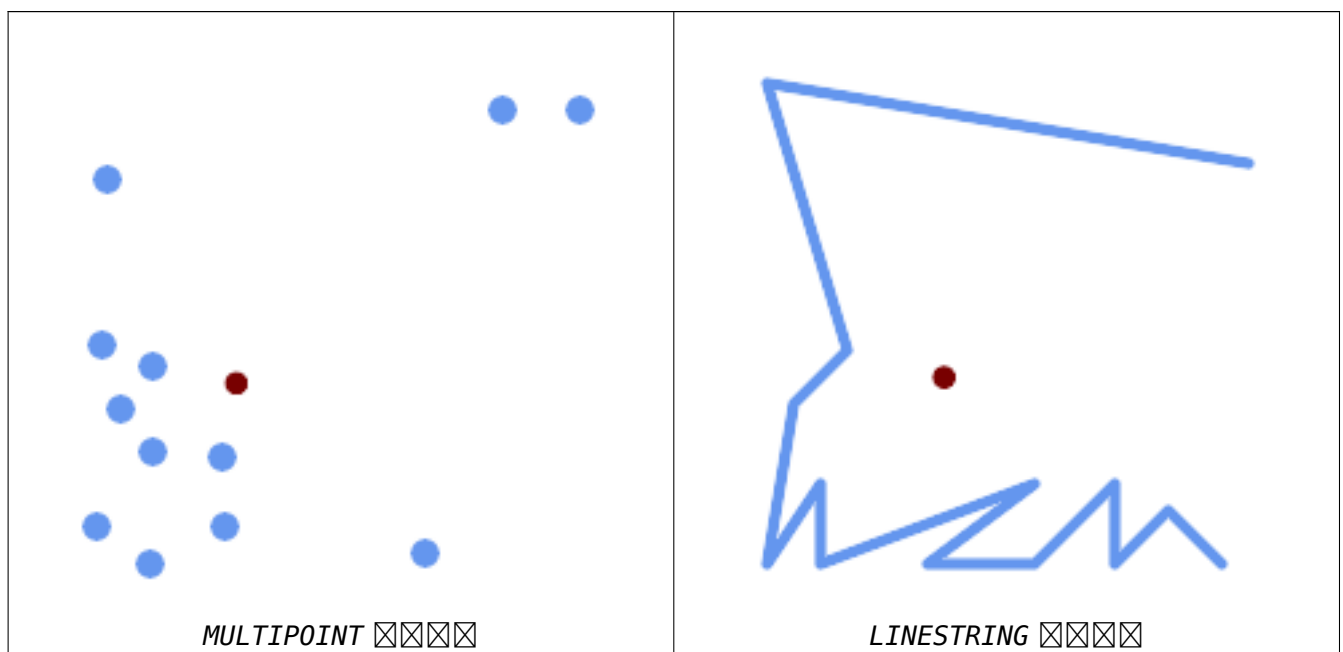
New in 2.3.0 : supports CIRCULARSTRING and COMPOUNDCURVE (using CurveToLine)

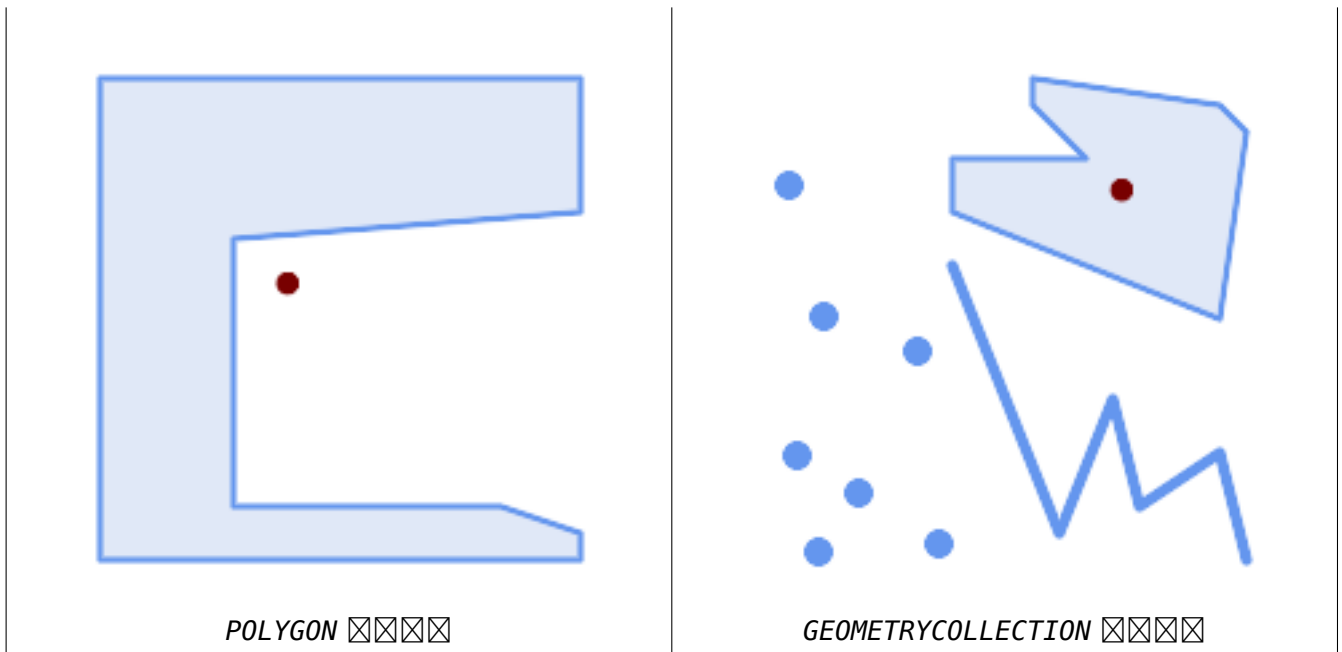
Availability: 2.4.0 support for geography was introduced.

- ✔ 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' ' ←

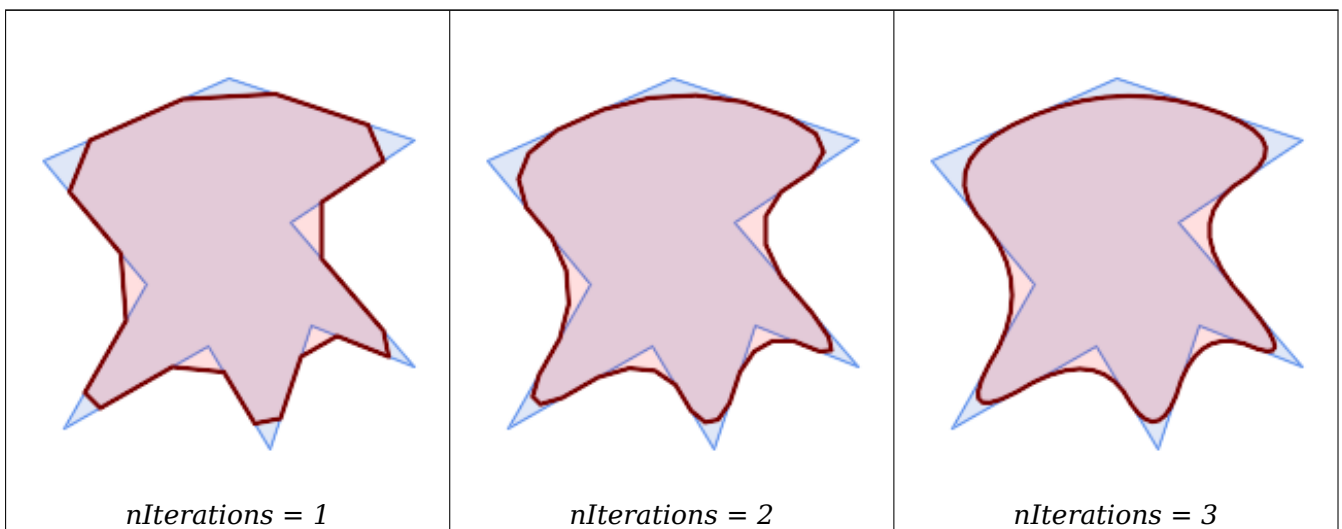
b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' ' ←

'-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' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' ' ←

POLYGON((2 2,6 6,6 10,2 14,0 12,0 4,2 2))

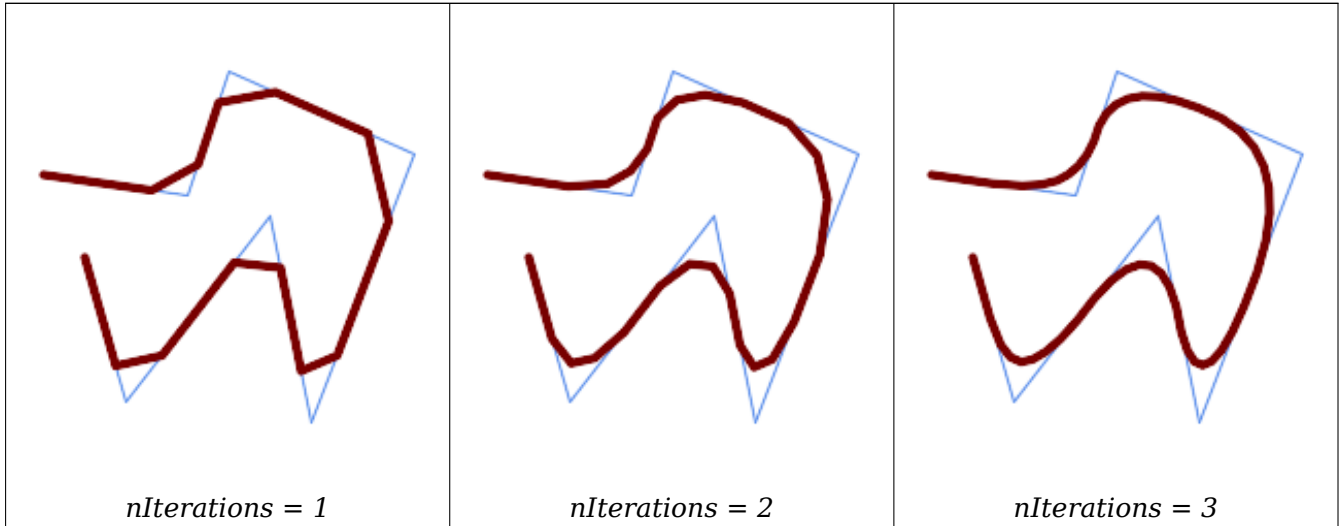
Smoothing a Polygon using 1, 2 and 3 iterations:



```
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.

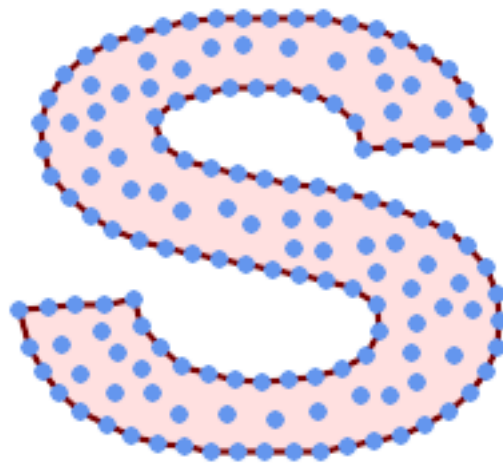
For point and linear inputs, the hull will enclose all the points of the inputs. For polygonal inputs, the hull will enclose all the points of the input *and also* all the areas covered by the input. If you want a point-wise hull of a polygonal input, convert it to points first using [ST_Points](#).

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 国国国国国国国国国国国国.

Enhanced: 3.3.0, GEOS native implementation enabled for GEOS 3.11+

国国



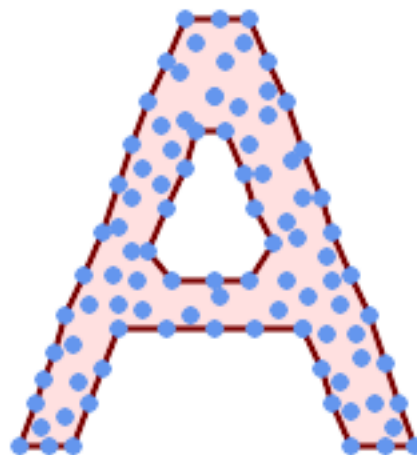
Concave Hull of a MultiPoint

```
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

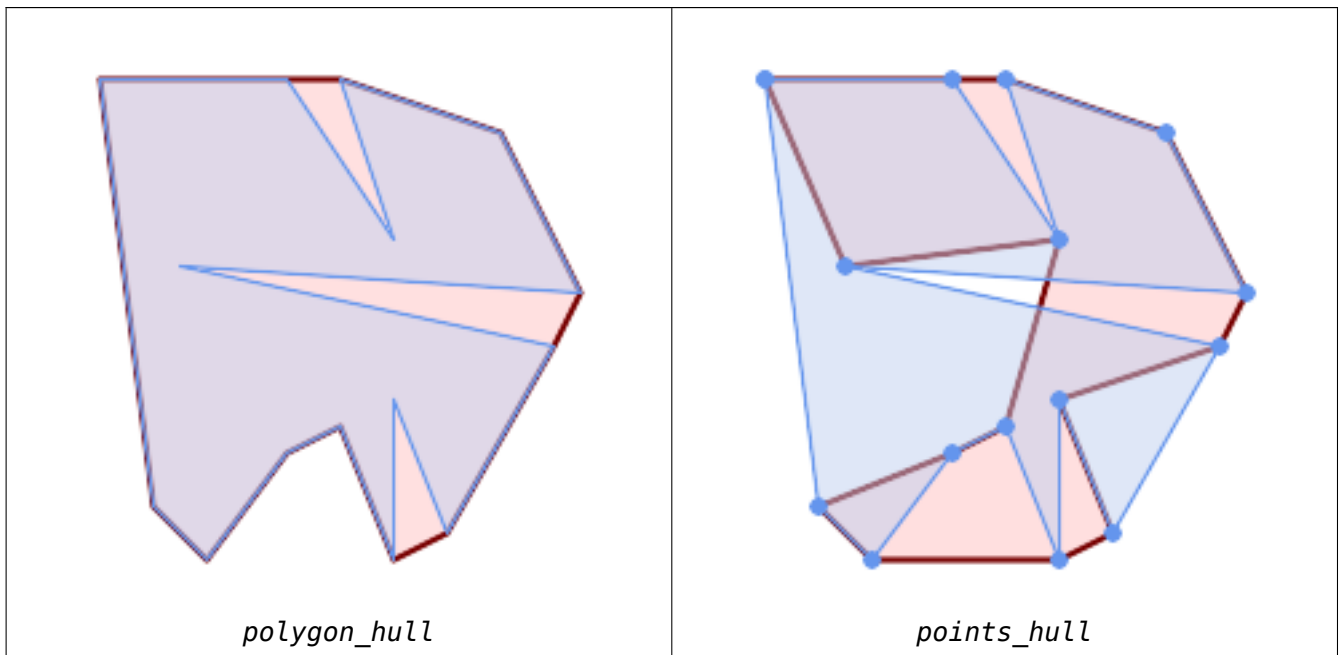
```

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);

国国

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.

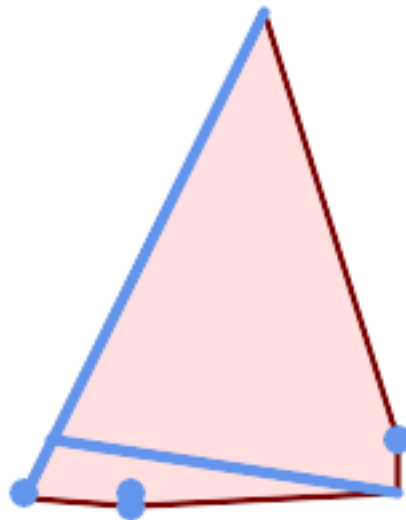
In the general case the convex hull is a Polygon. The convex hull of two or more collinear points is a two-point LineString. The convex hull of one or more identical points is a Point.

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 国国国国国

- ✔ 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.

国国



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;
```

☐☐

[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);

☐☐

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:

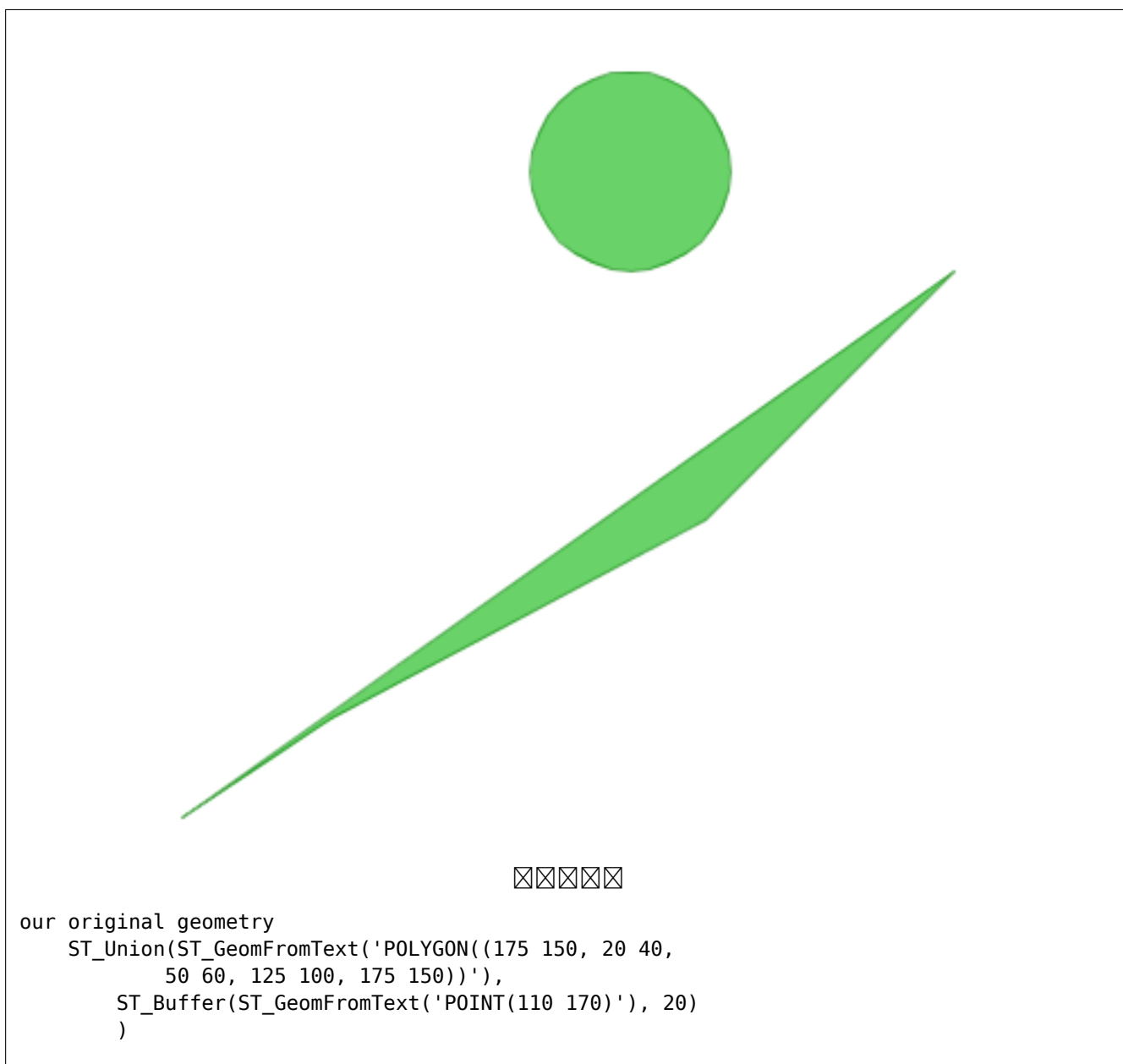
- 0 - a GEOMETRYCOLLECTION of triangular POLYGONS (default)
- 1 - a MULTILINESTRING of the edges of the triangulation
- 2 - A TIN of the triangulation

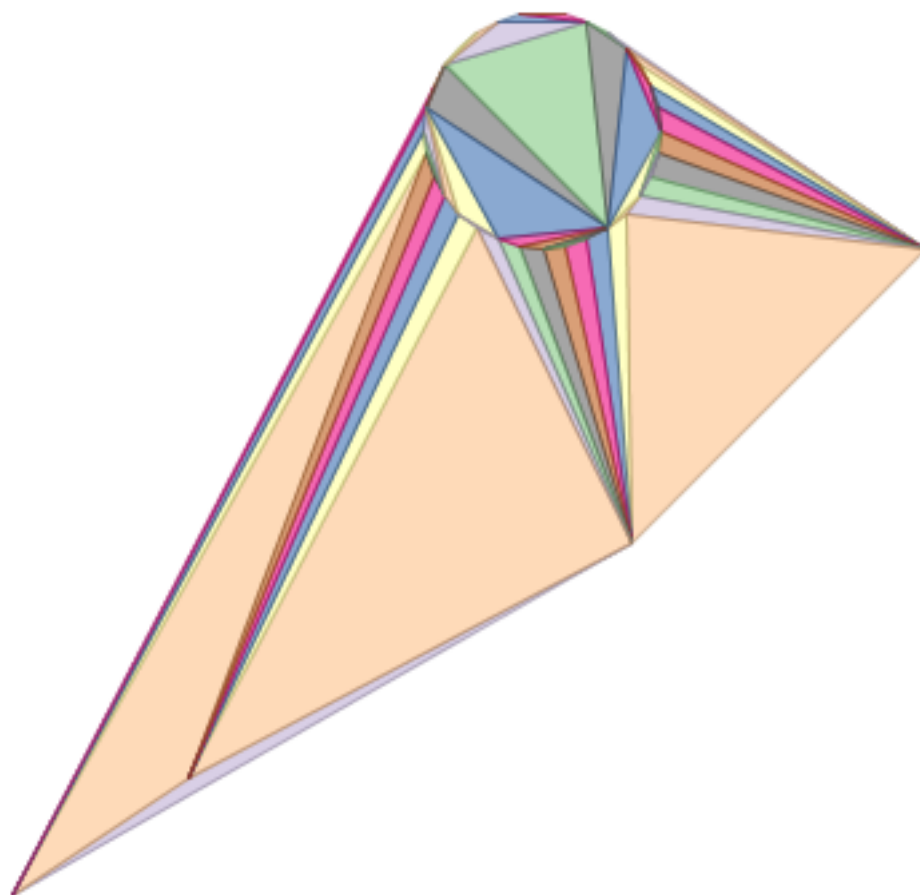
GEOS ☐☐☐☐☐

2.1.0 ☐☐☐☐☐☐☐☐☐☐.

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

☐☐

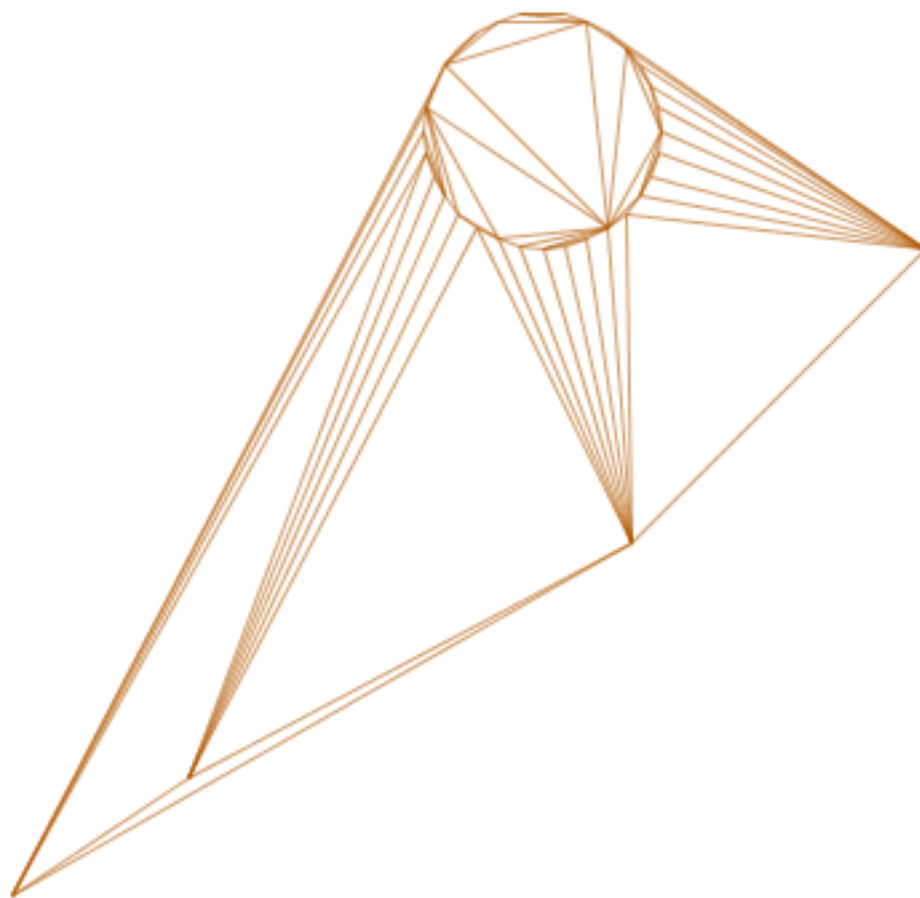




ST DelaunayTriangles:

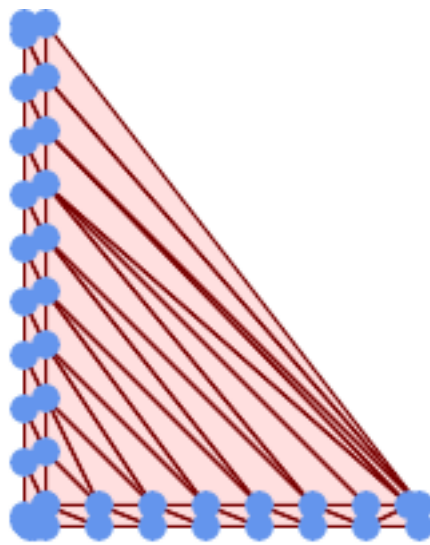
geometries overlaid multilinestring triangles

```
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;
```



```
-- 繁體中文 55 繁體中文 45 繁體中文
```

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);

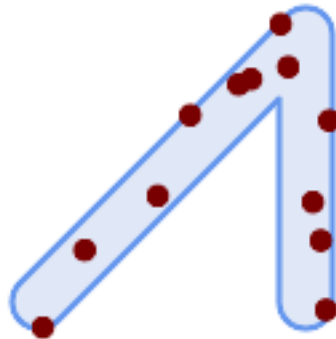
☐☐

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 简体中文.

Enhanced: 3.0.0, added seed parameter

图例



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;
```

图例

ST_DumpPoints

7.14.10 ST_GeometricMedian

ST_GeometricMedian — 返回几何体的几何中位数 (median) 几何体。

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.

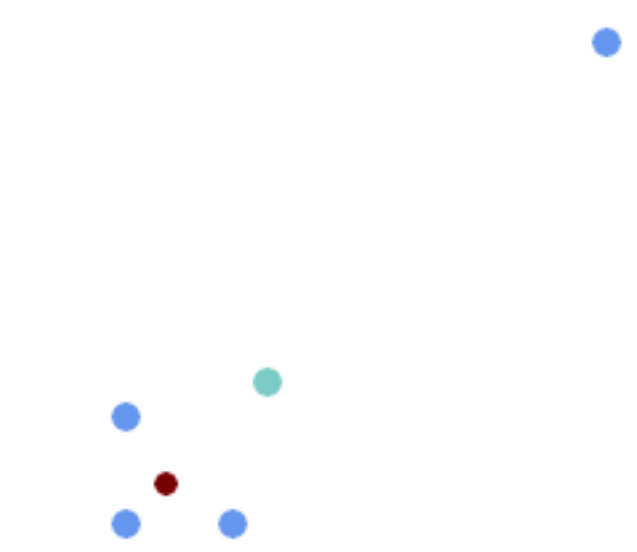
If present, the input point M values are interpreted as their relative weights.

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 简体中文

Enhanced: 3.3.0 accept a directed parameter.

Requires GEOS >= 3.11.0 to use the directed parameter.

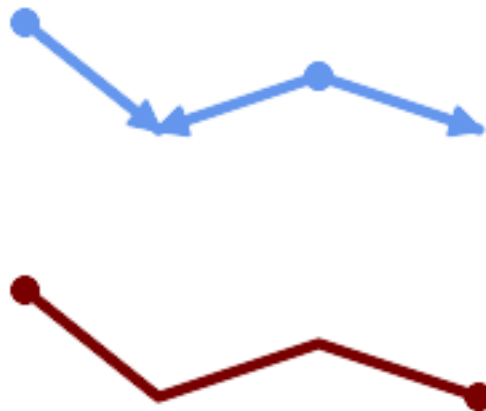
1.1.0 简体中文.



Warning

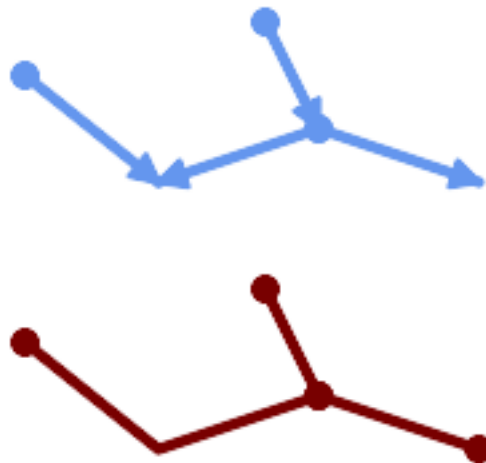
This function strips the M dimension.

图例



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.

ST_MaximumInscribedCircle

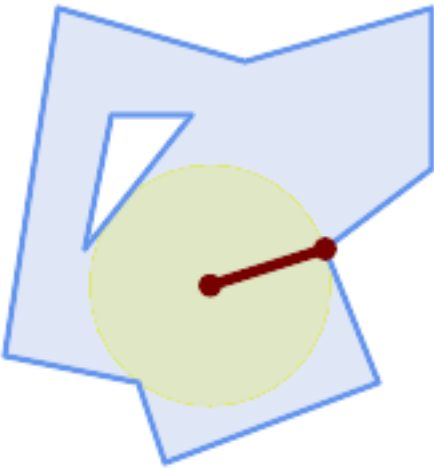
Finds the largest circle that is contained within a (multi)polygon, or which does not overlap any lines and points. Returns a record with fields:

- 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.

Diagram

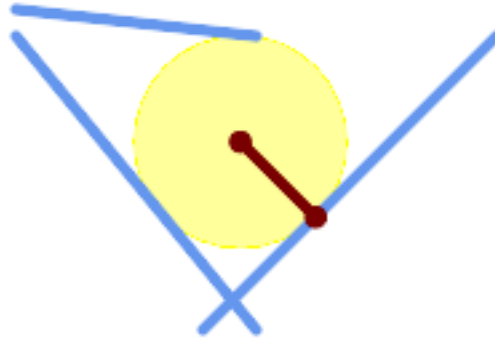


Maximum inscribed circle of a polygon. Center, nearest point, and radius are returned.

```
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

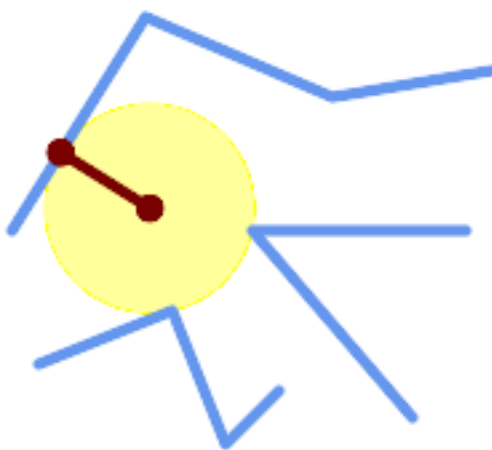
To find the largest empty circle in the interior of a polygon, see [ST_MaximumInscribedCircle](#).

Availability: 3.4.0.

Requires GEOS >= 3.9.0.

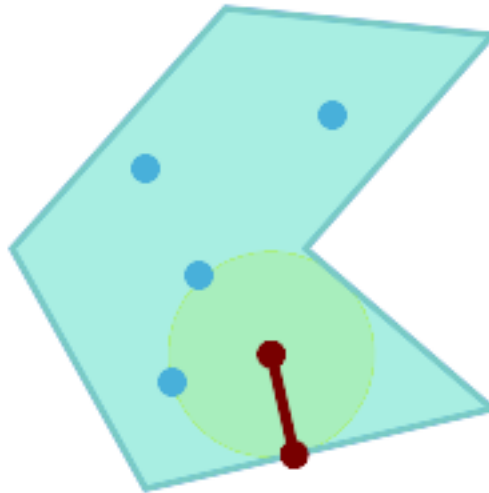
☒☒

```
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
  );
```



7.14.14 ST_MinimumBoundingCircle

Synopsis

[illegible]

1.4.0 ☒☒☒☒☒☒☒☒☒☒☒.

图 10

```
SELECT d.disease_type,
       ST_MinimumBoundingCircle(ST_Collect(d.geom)) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```

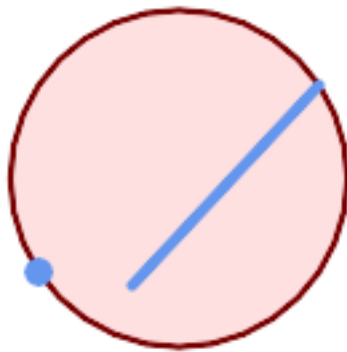


图 11: 使用 ST_MinimumBoundingCircle 函数计算最小包围圆的 SQL 查询结果。

```
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))
```

图 12

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);

返回

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 简体中文。

返回

```
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

返回

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);

返回

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.

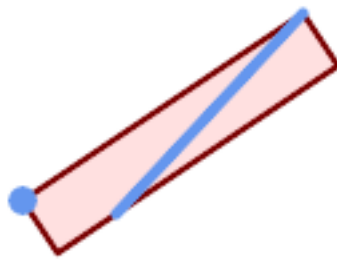
Availability: 2.5.0.

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=“);

ST_OffsetCurve

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.

ST_OffsetCurve(geometry, distance, [options]).

Note that output may be a MULTILINESTRING or EMPTY for some jigsaw-shaped input geometries.

ST_OffsetCurve(geometry, distance, 'quad_segs=8 join=round mitre_limit=5') = ST_MakeValid(ST_OffsetCurve(geometry, distance, 'quad_segs=8 join=round')):

- 'quad_segs=#': 整数 (quarter circle) 的段数 (默认 8)
- 'join=round|mitre|bevel': 连接方式 ("round" (round)). 'mitre' (mitre) 'bevel' (bevel)
- 'mitre_limit=#.#': 限制值 (默认 5.0). 'mitre_limit' 和 'miter_limit' 是同义词。

GEOS 兼容性

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 兼容性。

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.

ST_MakeValid

ST_MakeValid(geometry).

```
SELECT ST_Union(
  ST_OffsetCurve(f.geom, f.width/2, 'quad_segs=4 join=round'),
  ST_OffsetCurve(f.geom, -f.width/2, 'quad_segs=4 join=round')
) as track
FROM someroadstable;
```



15, 'quad_segs=4 join=round'
15

```
SELECT ST_AsText(ST_OffsetCurve(↵
  ST_GeomFromText(↵
    'LINESTRING(164 16,144 16,124 16,104 ↵
      16,84 16,64 16,↵
      44 16,24 16,20 16,18 16,17 17,↵
      16 18,16 20,16 40,16 60,16 80,16 100,↵
      16 120,16 140,16 160,16 180,16 195)')↵
    ,↵
    15, 'quad_segs=4 join=round'));
```

output

```
LINESTRING(164 1,18 1,12.2597485145237 ↵
  2.1418070123307,↵
  7.39339828220179 5.39339828220179,↵
  5.39339828220179 7.39339828220179,↵
  2.14180701233067 12.2597485145237,1 ↵
  18,1 195)
```

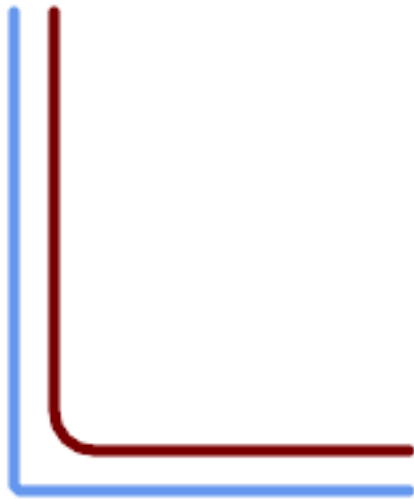


-15, 'quad_segs=4
join=round' -15

```
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

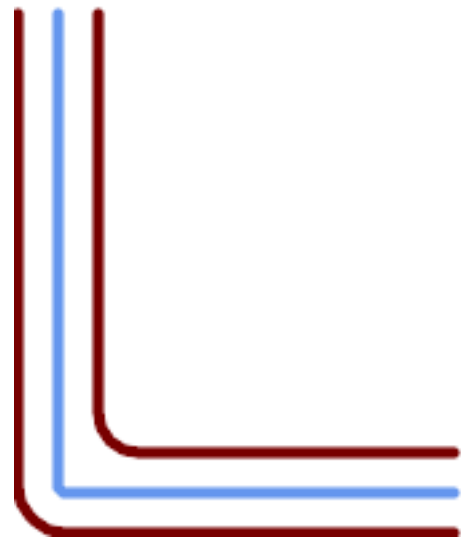
```
LINESTRING(31 195,31 31,164 31)
```



偏移距离为负值时，曲线将向左侧偏移。例如， $-30 + 15 = -15$ 表示向左偏移 15 个单位。

```
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)
```

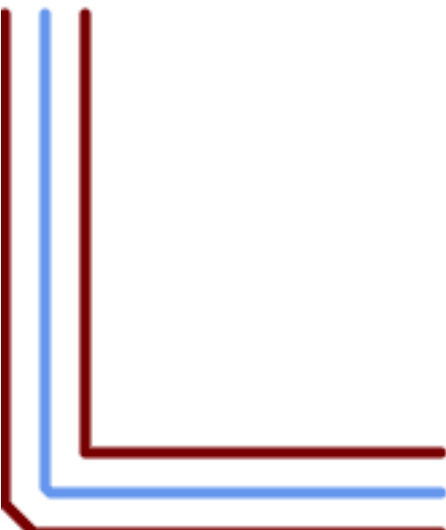


偏移距离为正值时，曲线将向右侧偏移。例如， 15 表示向右偏移 15 个单位。

```
SELECT ST_AsText(ST_Collect(
  ST_OffsetCurve(geom, 15, 'quad_segs=4
  join=round'),
  ST_OffsetCurve(ST_OffsetCurve(geom,
    -30, 'quad_segs=4 join=round'), -15,
    'quad_segs=4 join=round')
) ) As parallel_curves
FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104
16,84 16,64 16,
44 16,24 16,20 16,18 16,17 17,
16 18,16 20,16 40,16 60,16 80,16 100,
16 120,16 140,16 160,16 180,16 195)')
As geom;
```

parallel curves

```
MULTILINESTRING((164 1,18
1,12.2597485145237 2.1418070123307,
7.39339828220179
5.39339828220179,5.39339828220179 7.39339828220179,
2.14180701233067 12.2597485145237,1 18,1
195),
(164 31,46 31,40.2597485145236
32.1418070123307,35.3933982822018 35.3933982822018,
32.1418070123307 40.2597485145237,31
46,31 195))
```


 15, 'quad_segs=4 join=round' <pre>SELECT ST_AsText(ST_OffsetCurve(↵ ST_GeomFromText(↵ 'LINESTRING(164 16,144 16,124 16,104 ↵ 16,84 16,64 16,↵ 44 16,24 16,20 16,18 16,17 17,↵ 16 18,16 20,16 40,16 60,16 80,16 100,↵ 16 120,16 140,16 160,16 180,16 195)')↵ ,↵ 15, 'quad_segs=4 join=bevel'));</pre> output <pre>LINESTRING(164 1,18 1,7.39339828220179 ↵ 5.39339828220179,↵ 5.39339828220179 7.39339828220179,1 ↵ 18,1 195)</pre>	 15, -15. join=mitre mitre_limit=2.1 <pre>SELECT ST_AsText(ST_Collect(↵ ST_OffsetCurve(geom, 15, 'quad_segs=4 ↵ join=mitre mitre_limit=2.2'),↵ ST_OffsetCurve(geom, -15, 'quad_segs ↵ =4 join=mitre mitre_limit=2.2')↵))↵ FROM ST_GeomFromText(↵ 'LINESTRING(164 16,144 16,124 16,104 ↵ 16,84 16,64 16,↵ 44 16,24 16,20 16,18 16,17 17,↵ 16 18,16 20,16 40,16 60,16 80,16 100,↵ 16 120,16 140,16 160,16 180,16 195)')↵ As geom;</pre> output <pre>MULTILINESTRING((164 1,11.7867965644036 ↵ 1,1 11.7867965644036,1 195),↵ (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.



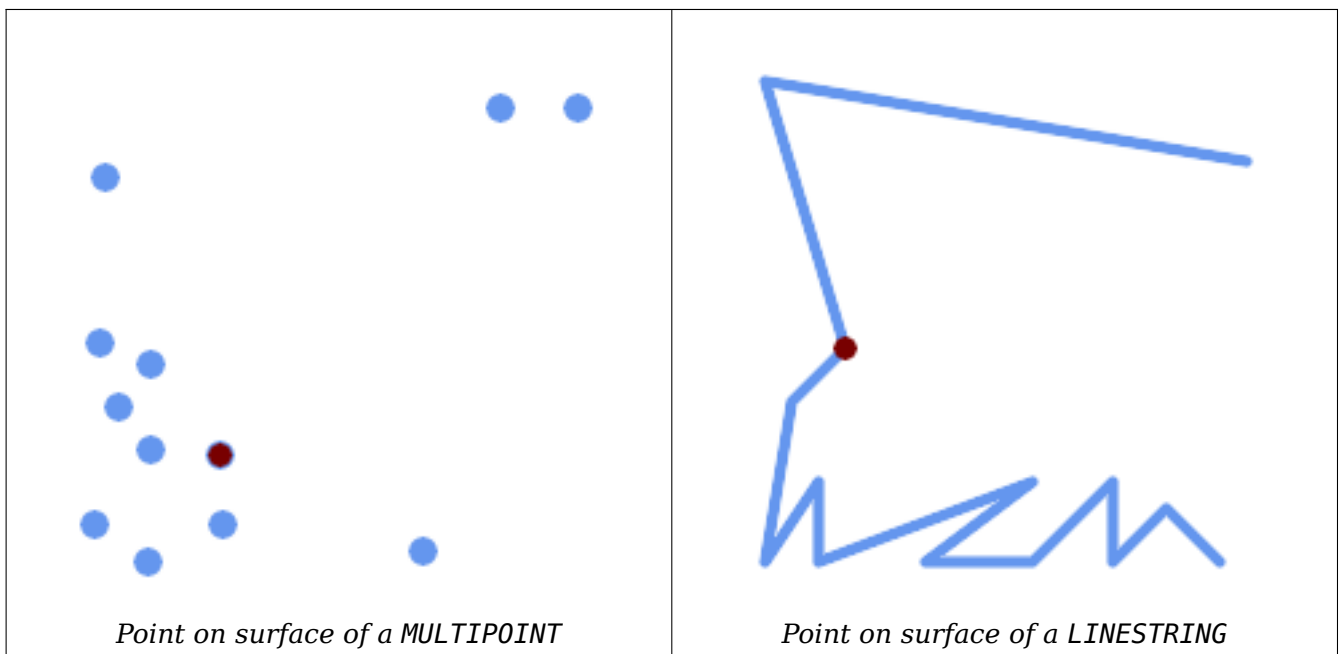
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.14.2 // s3.2.18.2

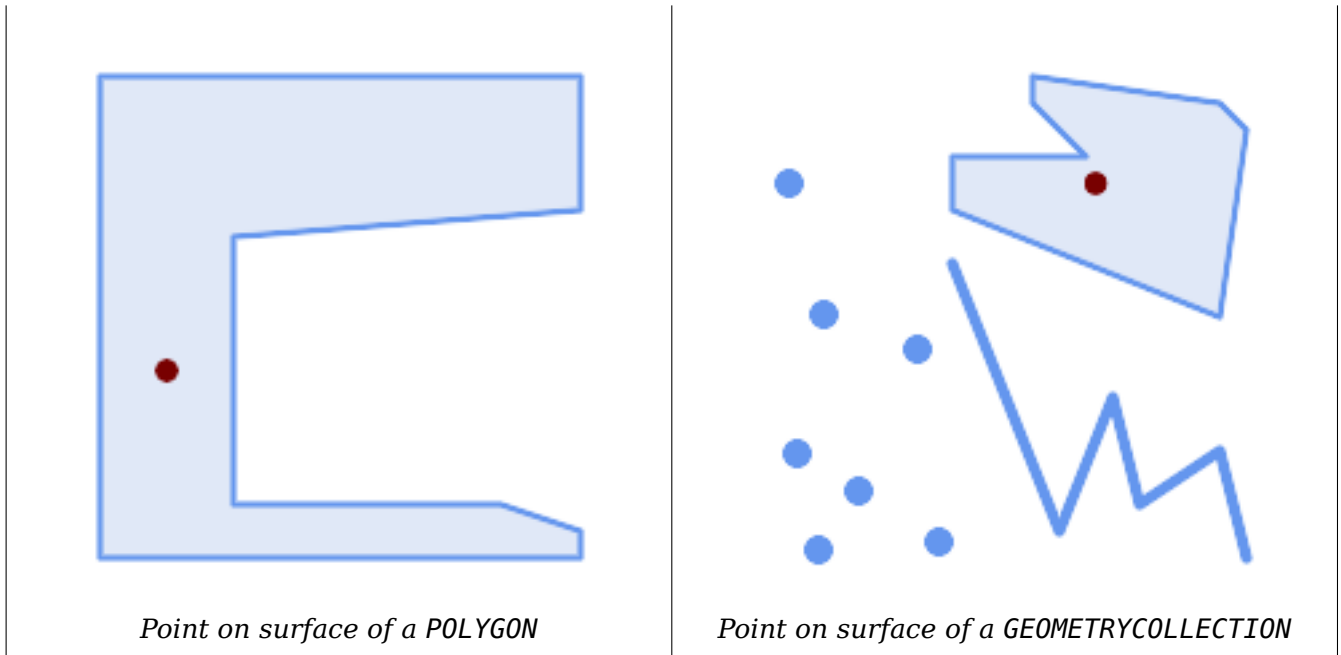


This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. The specifications define `ST_PointOnSurface` for surface geometries only. PostGIS extends the function to support all common geometry types. Other databases (Oracle, DB2, ArcSDE) seem to support this function only for surfaces. SQL Server 2008 supports all common geometry types.



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





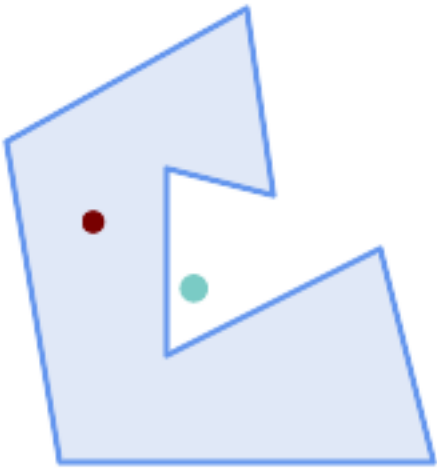
```
SELECT ST_AsText(ST_PointOnSurface('POINT(0 5) '::geometry));
-----
POINT(0 5)

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.

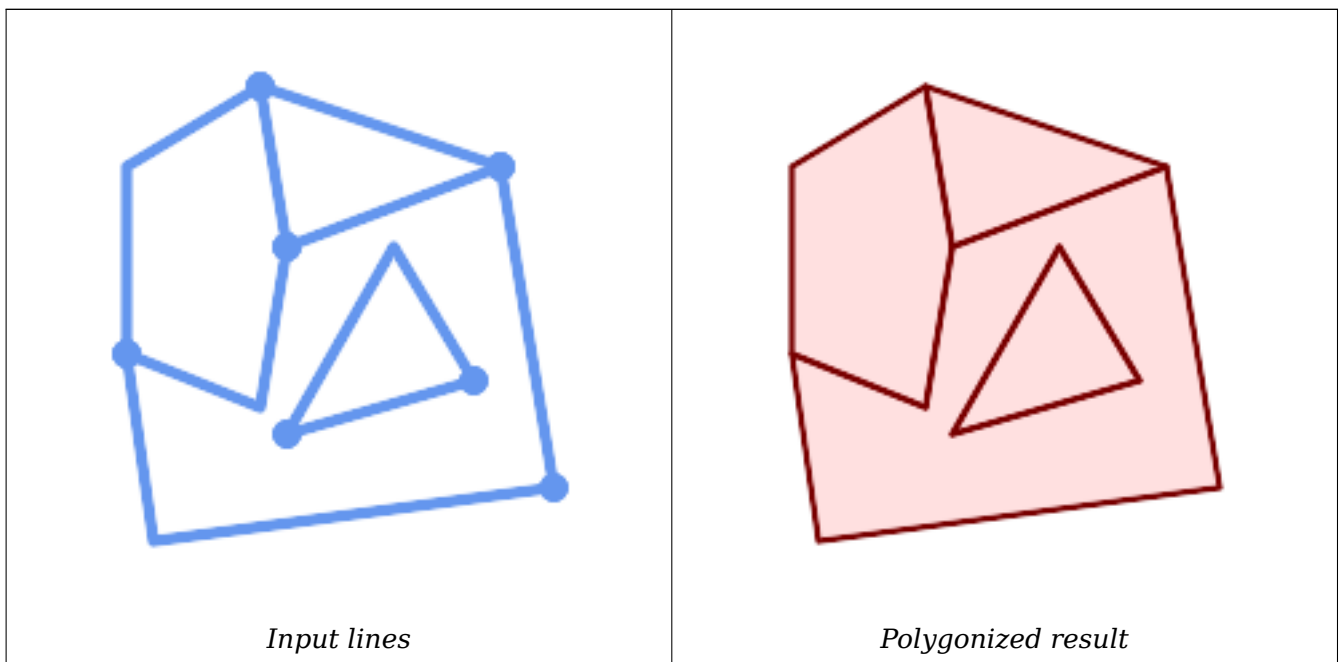
**Note**

GeometryCollections can be difficult to handle with external tools. Use **ST_Dump** to convert the polygonized result into separate polygons.

GEOS 3.10.2

1.0.0RC1 简体中文.

图



```
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.

图例

```
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)))
```

图例

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);

图例

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。如果两个几何体共享的路径是一个点，则返回该点。如果两个几何体共享的路径是一个线，则返回该线。

GEOS 3.10.0

2.0.0 版本。

图例: 共享路径


图 10-10 共享路径

图 10-11 共享路径
<pre>SELECT ST_AsText(ST_SharedPaths(ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125), (51 150,101 150,76 175,51 150))'), ST_GeomFromText('LINESTRING(151 100,126 156.25,126 125,90 161, 76 175)'))) As wkt ----- wkt ----- GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125), (101 150,90 161),(90 161,76 175)),MULTILINESTRING EMPTY)</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 简化几何体。

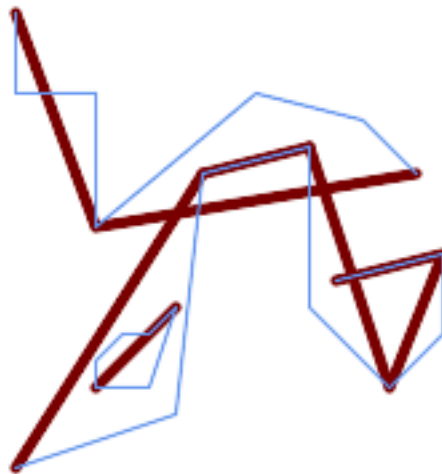
图 1.2.2

图 1.2.2 展示了使用 ST_Simplify 函数对几何体进行简化的结果。

```
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;
```

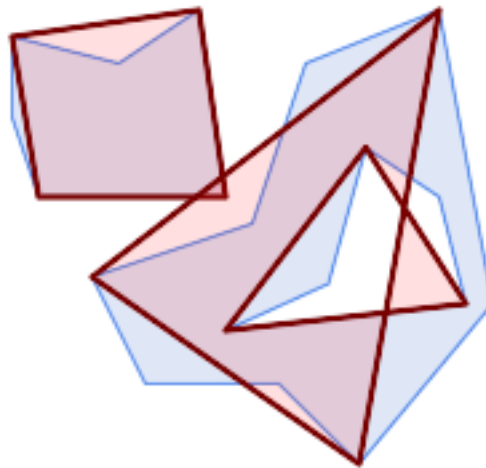
np_before	np01_notbadcircle	np05_notquitecircle	np1_octagon	np10_triangle	np100_geometrygoesaway
49	33	17	9	4	t

简化一组线条。线条在简化后可能会相交。



```
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);
```

简化一个 MultiPolygon。多边形结果可能是无效的。



```
SELECT ST_Simplify(
  '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_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 简体中文

1.3.3 简体中文.

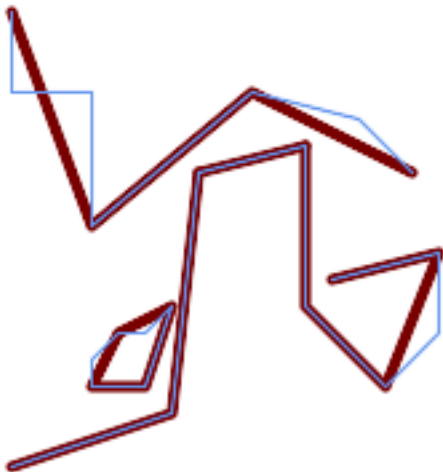
图

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;
```

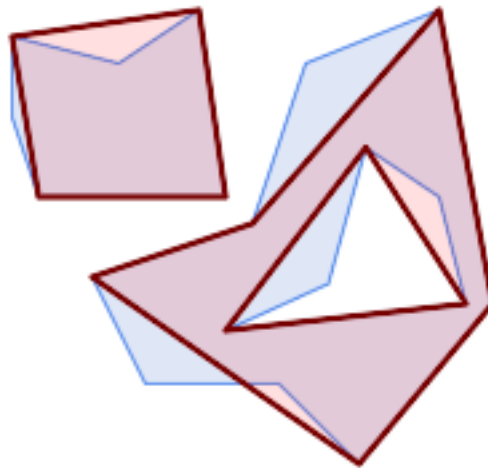
np_before	np01_notbadcircle	np05_notquitecircle	np1_octagon	np10_square	
49	33	17	9	5	
	5				

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  ←
    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_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.

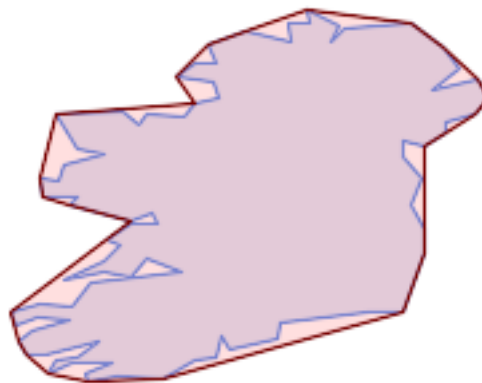
To get better results with geometries that contain relatively long line segments, it might be necessary to "segmentize" the input, as shown below.

GEOS 简体中文

Availability: 3.3.0.

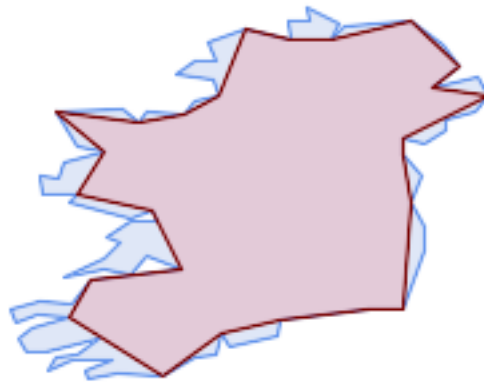
Requires GEOS >= 3.11.0.

图



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 , , .

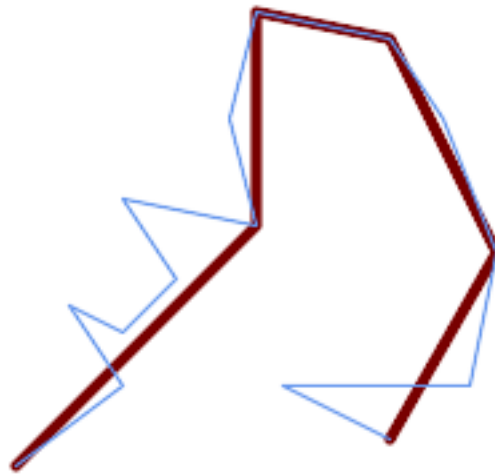
2.2.0 .

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;

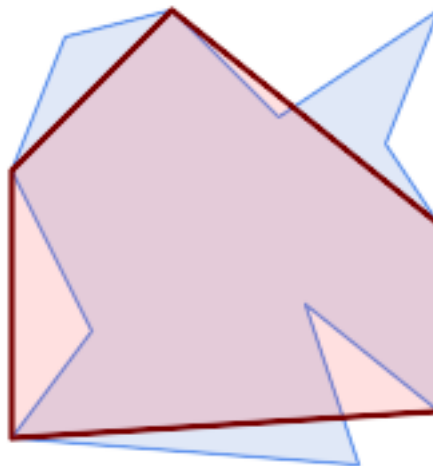
simplified
-----
LINESTRING(5 2,7 25,10 10)
```

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.

￼￼

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.

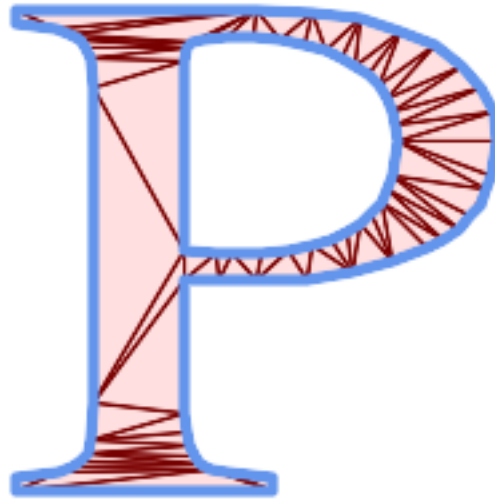
```
SELECT ST_AsText(
  ST_TriangulatePolygon('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))');

          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.

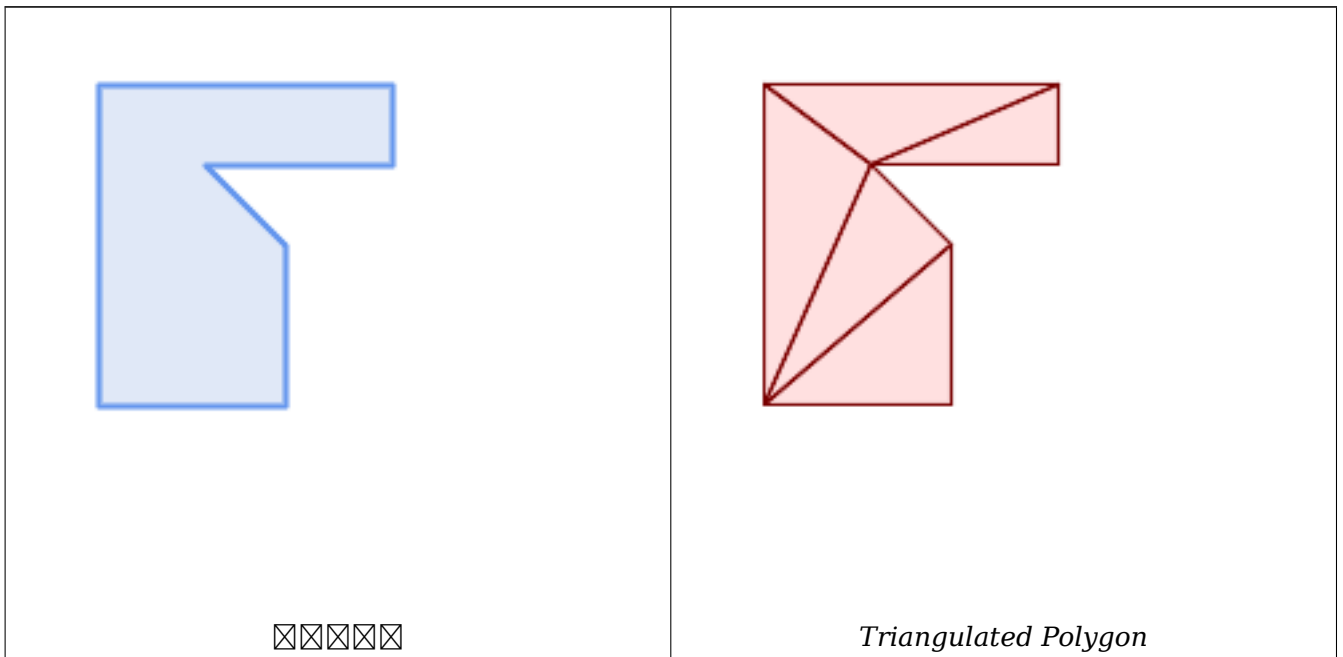
```
SELECT ST_AsText(ST_TriangulatePolygon(
  'POLYGON ((26 17, 31 19, 34 21, 37 24, 38 29, 39 43, 39 161, 38 172, 36 176, 34 179, 30 181, 25 183, 10 185, 10 190, 100 190, 121 189, 139 187, 154 182, 167 177, 177 169, 184 161, 189 152, 190 141, 188 128, 186 123, 184 117, 180 113, 176 108, 170 104, 164 101, 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 圖形:

```
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.

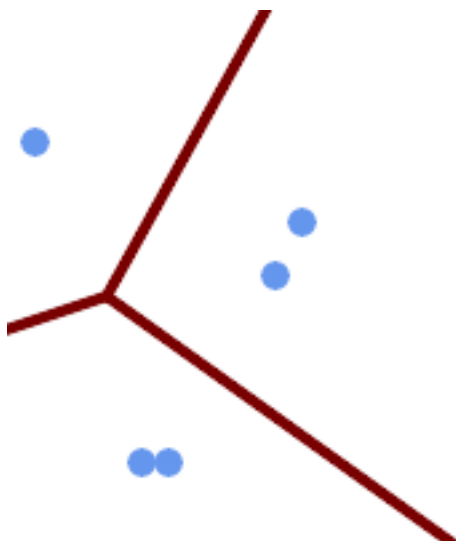
☐☐☐☐☐☐☐☐☐:

- **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))
```

图例

[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);

图例

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.

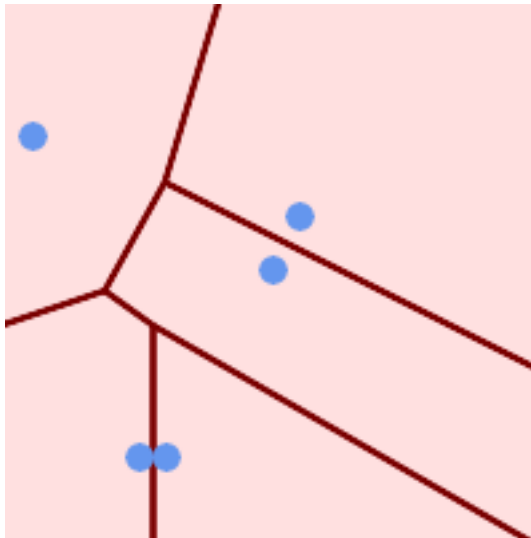
图例:

- **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 图例

2.3.0 图例.

图图

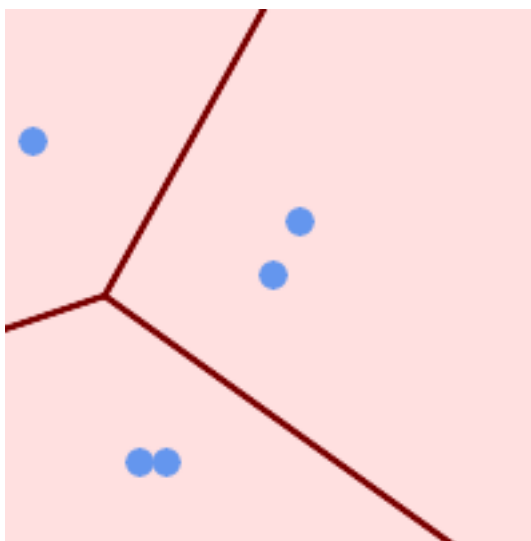


Points overlaid on top of Voronoi diagram

```
SELECT ST_VoronoiPolygons(
    'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120) '::geometry
) AS geom;
```

ST_AsText output

```
GEOMETRYCOLLECTION(POLYGON((-110 43.3333333333333,-110 270,100.5 270,59.3478260869565 ↵
    132.826086956522,36.8181818181818 92.2727272727273,-110 43.3333333333333)),
POLYGON((55 -90,-110 -90,-110 43.3333333333333,36.8181818181818 92.2727272727273,55 ↵
    79.2857142857143,55 -90)),
POLYGON((230 47.5,230 -20.7142857142857,55 79.2857142857143,36.8181818181818 ↵
    92.2727272727273,59.3478260869565 132.826086956522,230 47.5)),POLYGON((230 ↵
    -20.7142857142857,230 -90,55 -90,55 79.2857142857143,230 -20.7142857142857)),
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```



Voronoi diagram, with tolerance of 30 units

```
SELECT ST_VoronoiPolygons(  
    'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)::geometry,  
    30) AS geom;
```

ST_AsText output

```
GEOMETRYCOLLECTION(POLYGON((-110 43.333333333333,-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

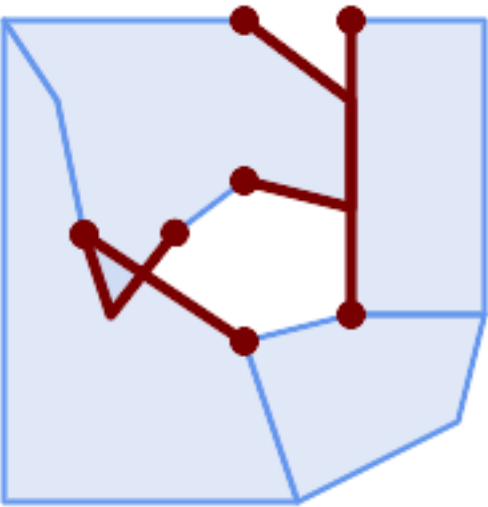
As a window function a value is returned for every input polygon. For polygons which violate one or more of the validity conditions the return value is a MULTILINESTRING containing the problematic edges. Coverage-valid polygons return the value NULL. Non-polygonal or empty geometries also produce NULL 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_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.

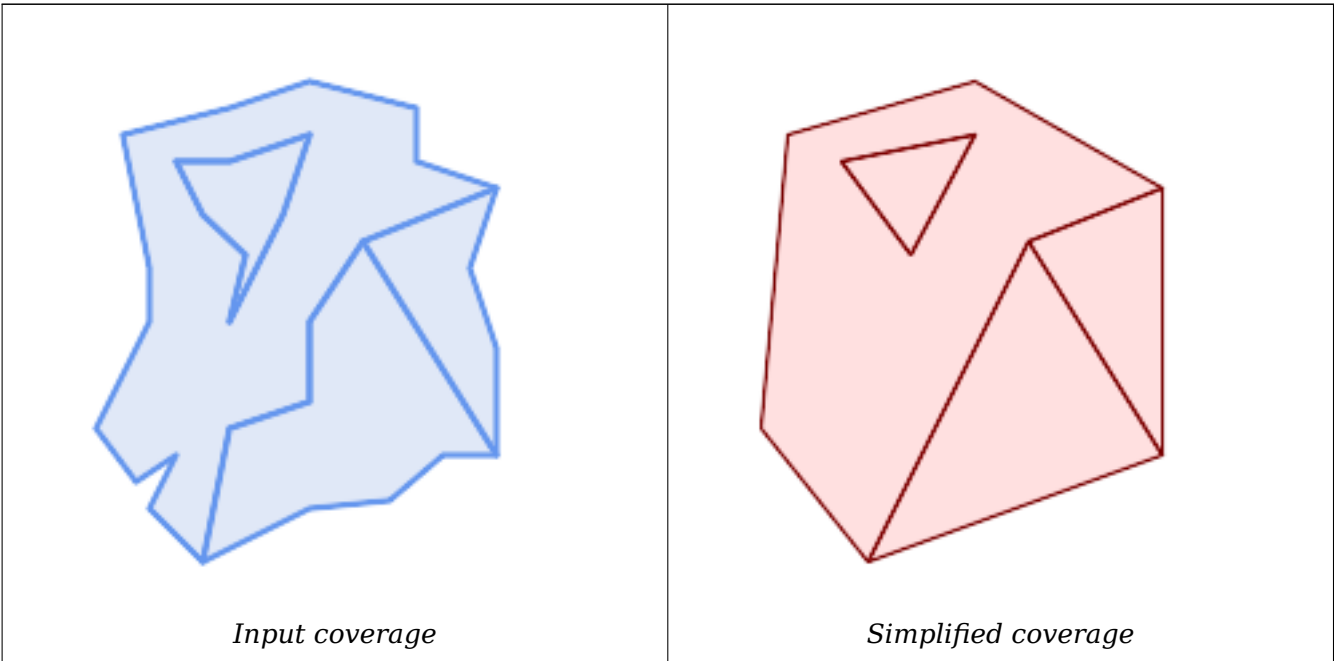


Note

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

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);



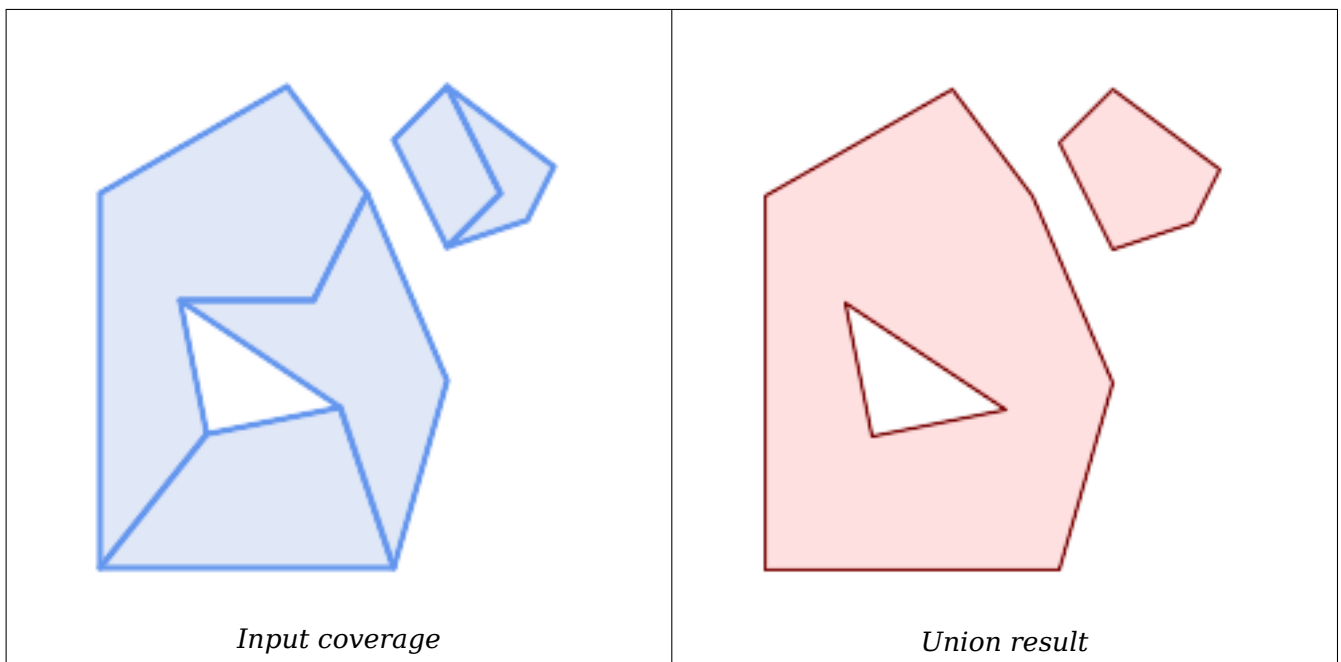
An aggregate function which unions a set of polygons forming a polygonal coverage. The result is a polygonal geometry covering the same area as the coverage. This function produces the same result as **ST_Union**, but uses the coverage structure to compute the union much faster.



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_AsBinary](#)

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);

ST_Affine

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.

PostGIS 2.0.0 引入, 支持 TIN 几何类型。

Availability: 1.1.2. Name changed from Affine to ST_Affine in 1.2.2

**Note**

1.3.4 版本 (curve) 几何类型。1.3.4 版本引入。



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.

SQL

```
--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;
-----+-----
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)
```

SQL

[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);
```

SQL

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).

History: 2.0.0 **Enhanced:** 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).

❏❏❏❏: 2.0.0 简体中文, ❏❏❏ TIN ❏❏❏❏❏❏❏❏❏.

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).

例

```
--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)
```

例

[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);

例

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

例: 2.0.0 支持, TIN 支持.



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);

返回

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).

兼容性: 2.0.0 兼容 PostgreSQL, 支持 TIN 数据类型。

Availability: 1.1.2. Name changed from RotateZ to ST_RotateZ in 1.2.2



Note

1.3.4 兼容 PostgreSQL (curve) 数据类型。1.3.4 兼容 PostgreSQL。

- ✓ 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).

返回

```
--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 简体中文 (curve) 简体中文. 1.3.4 简体中文.

Availability: 1.1.0.

☒☒☒: 2.0.0 简体中文, 简体中文 TIN 简体中文.

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.

￼￼

```
--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)
```

￼￼

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 deltaz);
```

￼￼

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 版本 (curve) 支持。1.3.4 版本支持。

1.2.2 版本。



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

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 简体中文 (curve) 简体中文. 1.3.4 简体中文.

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));
           st_asewkt
```

```
-----
LINESTRING(1.5 6 3,1.5 4 1)
```

```
--Buffer a point to get an approximation of a circle, convert to curve and then translate ↵
1,2 and scale it 3,4
```

```
SELECT ST_AsText(ST_Transscale(ST_LineToCurve(ST_Buffer('POINT(234 567)', 3)),1,2,3,4));
```

```
-----
CURVEPOLYGON(CIRCULARSTRING(714 2276,711.363961030679 2267.51471862576,705 ↵
2264,698.636038969321 2284.48528137424,714 2276))
```

⊠

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.

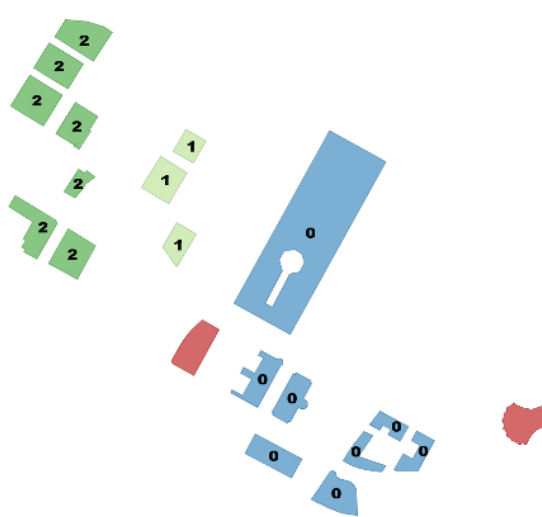
2.3.0 ￼￼￼￼￼￼￼￼￼.



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 简体中文.

⌘

```
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.

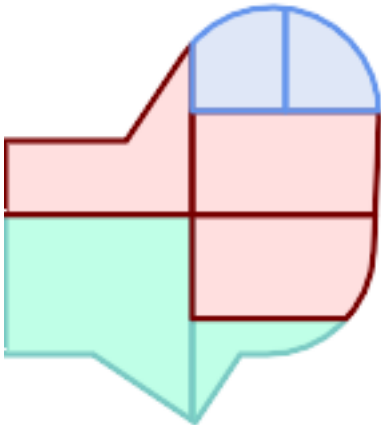
Enhanced: 3.2.0 Support for max_radius

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 clustering and weighting. Identify at least 20 regions based on **Kontur Population Data** that do not span more than 3000 km from their center:

```
create table kontur_population_3000km_clusters as
select
  geom,
  ST_ClusterKMeans(
    ST_Force4D(
      ST_Transform(ST_Force3D(geom), 4978), -- cluster in 3D XYZ CRS
      mvalue => population -- set clustering to be weighed by population
    ),
    20, -- aim to generate at least 20 clusters
    max_radius => 3000000 -- but generate more to make each under 3000 km radius
  ) over () as cid
from
  kontur_population;
```



World population clustered to above specs produces 46 clusters. Clusters are centered at well-populated regions (New York, Moscow). Greenland is one cluster. There are island clusters that span across the antimeridian. Cluster edges follow Earth's curvature.

￼

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 简体中文



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;
```

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_ClusterWithin](#), [ST_ClusterDBSCAN](#), [ST_ClusterIntersecting](#), [ST_ClusterIntersectingWin](#),

7.18 Bounding Box Functions

7.18.1 Box2D

Box2D — Returns a BOX2D representing the 2D extent of a geometry.

Synopsis

box2d **Box2D**(geometry geom);

❏❏

Returns a **box2d** representing the 2D extent of the geometry.

❏❏❏❏: 2.0.0 [❏❏❏❏❏❏❏❏❏❏](#), [❏❏❏❏](#) TIN [❏❏❏❏❏❏❏❏❏❏❏❏](#).

- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

❏

```
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)
```

❏

Box3D, ST_GeomFromText

7.18.2 Box3D

Box3D — Returns a BOX3D representing the 3D extent of a geometry.

Synopsis

box3d **Box3D**(geometry geom);

❏

Returns a **box3d** representing the 3D extent of the geometry.

❏❏❏: 2.0.0 ❏❏❏❏❏❏❏, ❏❏❏ TIN ❏❏❏❏❏❏❏❏❏.

- ✅ 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 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);
--10 meter expanded 3D box of a 3D box
SELECT ST_Expand(CAST('BOX3D(778783 2951741 1,794875 2970042.61545891 10)' As box3d),10)
--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));
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.

`box3d`: 2.0.0 `box3d`, `TIN`.

- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

`box3d`



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))
```

`box3d`

`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 **ST_3DExtent**, **ST_3DExtent** TIN **ST_3DExtent**.

Changed: 2.0.0 In prior versions this used to be called ST_Extent3D

- ✓ This function supports 3d and will not drop the z-index.
- ✓ This method supports Circular Strings and Curves.
- ✓ This function supports Polyhedral surfaces.
- ✓ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_MakePoint(x,y,z) As geom
      FROM generate_series(1,3) As x
      CROSS JOIN generate_series(1,2) As y
      CROSS JOIN generate_series(0,2) As Z) As foo;
      b3extent
-----
BOX3D(1 1 0,3 2 2)

--Get the extent of various elevated circular strings
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_Translate(ST_Force_3DZ(ST_LineToCurve(ST_Buffer(ST_Point(x,y),1))),0,0,z)
      As geom
      FROM generate_series(1,3) As x
      CROSS JOIN generate_series(1,2) As y
      CROSS JOIN generate_series(0,2) As Z) As foo;
      b3extent
-----
BOX3D(1 0 0,4 2 2)
```

See Also

ST_Extent, **ST_Force3DZ**, **ST_SetSRID**

7.18.7 ST_MakeBox2D

ST_MakeBox2D — Creates a BOX2D defined by two 2D point geometries.

Synopsis

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

☒☒

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
```

返回

`ST_GeomFromEWKT`, `ST_GeomFromText`, `ST_XMin`, `ST_XMax`, `ST_YMax`, `ST_YMin`, `ST_ZMax`

7.19 线性引用 (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);

返回

返回一个点，该点位于沿输入线串的给定分数位置处。该函数返回的点的类型与输入线串的类型相同。如果输入线串是 2D 的，则返回的点是 2D 的。如果输入线串是 3D 的，则返回的点是 3D 的。如果输入线串是 geography 类型的，则返回的点是 geography 类型的。该函数的输入参数 `a_fraction` 必须在 0 和 1 之间，并且是一个 Float8 类型的值。

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。如果输入线串的 `a_fraction` 为 0.0，则返回的点是线串的起始点。

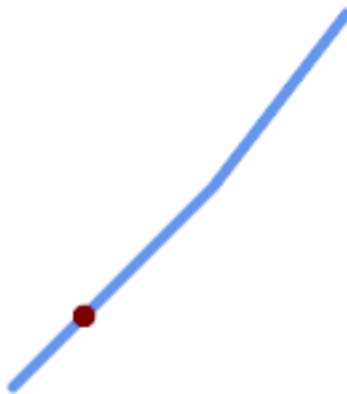
0.8.2 版本中，该函数返回的点是 2D 的。1.1.1 版本中，该函数返回的点是 3D 的。

兼容性: 2.1.0 版本中，该函数名为 `ST_Line_Interpolate_Point`。



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

图例



20% 图例 (0.20) 沿线的点

-- 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)
```

图例

[ST_LineInterpolatePoints](#), [ST_LineInterpolatePoint](#), [ST_LineMerge](#)

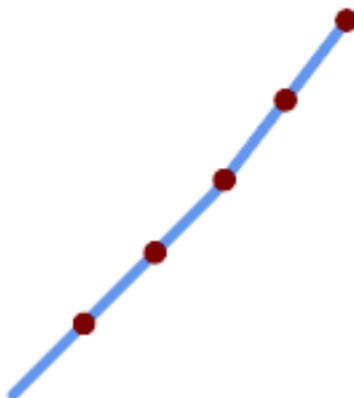
7.19.2 ST_3DLineInterpolatePoint

ST_3DLineInterpolatePoint — Returns a point interpolated along a 3D line at a fractional location.

Synopsis

geometry **ST_3DLineInterpolatePoint**(geometry a_linestring, float8 a_fraction);

图 10



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))
```

图 11

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);
```

图 12

图 12 展示了 ST_LineLocatePoint 函数的使用。图中显示了一个点（红色圆点）和一条线（蓝色线）。函数返回了该点到线的最近点的分数位置。图中还显示了 ST_LineInterpolatePoint 函数的使用，它返回了线上指定分数位置的点。图中还显示了 ST_LineSubstring 函数的使用，它返回了线上指定分数位置之间的子字符串。

图 12 展示了 ST_LineLocatePoint 函数的使用。图中显示了一个点（红色圆点）和一条线（蓝色线）。函数返回了该点到线的最近点的分数位置。图中还显示了 ST_LineInterpolatePoint 函数的使用，它返回了线上指定分数位置的点。图中还显示了 ST_LineSubstring 函数的使用，它返回了线上指定分数位置之间的子字符串。

图 12 展示了 ST_LineLocatePoint 函数的使用。图中显示了一个点（红色圆点）和一条线（蓝色线）。函数返回了该点到线的最近点的分数位置。图中还显示了 ST_LineInterpolatePoint 函数的使用，它返回了线上指定分数位置的点。图中还显示了 ST_LineSubstring 函数的使用，它返回了线上指定分数位置之间的子字符串。

1.1.0 版本中首次引入。

更新记录：2.1.0 版本中，在 2.0.x 版本中引入 ST_Line_Locate_Point 函数。

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 在 2D 模式下，Z 索引 (如果有) 将被忽略。在 3D 模式下，Z 索引将被保留。

Enhanced: 3.4.0 - Support for geography was introduced.

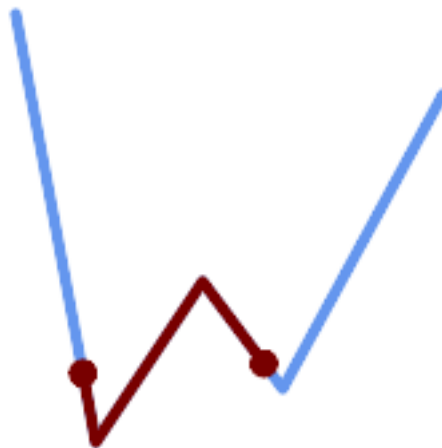
更新: 2.1.0 版本, 在 2.0.x 版本中引入 **ST_Line_Substring** 函数。

1.1.0 版本引入。1.1.1 版本 Z 索引支持。



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

图例



图例 1/3 在 (0.333, 0.666) 处切割

```
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	geom_sub
LINESTRING(-104.167064 38.854691,-87.674646 41.849854)	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 简体中文 ST_Locate_Along_Measure 简体中文.

简体中文: 2.0.0 简体中文 ST_Locate_Along_Measure 简体中文. 简体中文, 简体中文.



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(
    'MULTILINESTRING((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 offset = 0);

注意

ST_Locate_Between_Measures 返回的几何体可能包含无效的几何体。例如，如果输入几何体是一个非凸多边形，那么返回的几何体可能是一个无效的几何体。在这种情况下，ST_Locate_Between_Measures 返回的几何体可能是一个无效的几何体。在这种情况下，ST_Locate_Between_Measures 返回的几何体可能是一个无效的几何体。

Clipping a non-convex POLYGON may produce invalid geometry.

The semantic is specified by the *ISO/IEC 13249-3 SQL/MM Spatial* standard.

1.1.0 添加 ST_Locate_Between_Measures 函数。

注意: 2.0.0 添加 ST_Locate_Along_Measure 函数。该函数返回一个几何体，该几何体是输入几何体的一部分，该部分位于指定的测量值之间。

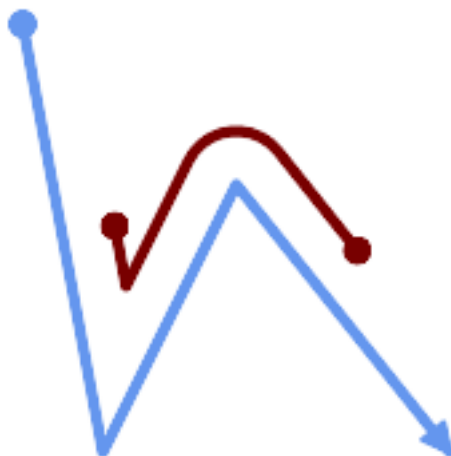
Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE.

✓ This function supports M coordinates.

✓ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

注意

```
SELECT ST_AsText(
  ST_LocateBetween(
    'MULTILINESTRING M ((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))':: geometry,
    1.5, 3 ));
-----
GEOMETRYCOLLECTION M (LINESTRING M (1 2 3,3 4 2,9 4 3),POINT M (1 2 3))
```



A LineString with the section between measures 2 and 8, offset to the left

```
SELECT ST_AsText( ST_LocateBetween(
  ST_AddMeasure('LINESTRING (20 180, 50 20, 100 120, 180 20)', 0, 10),
  2, 8,
  20
));
-----
```

```
MULTILINESTRING((54.49835019899045 104.53426957938231,58.70056060327303  ←
82.12248075654186,69.16695286779743 103.05526528559065,82.11145618000168  ←
128.94427190999915,84.24893681714357 132.32493442618113,87.01636951231555  ←
135.21267035596549,90.30307285299679 137.49198684843182,93.97759758337769  ←
139.07172433557758,97.89298381958797 139.8887023914453,101.89263860095893  ←
139.9102465862721,105.81659870902816 139.13549527600819,109.50792827749828  ←
137.5954340631298,112.81899532549731 135.351656550512,115.6173761888606  ←
132.49390095108848,145.31017306064817 95.37790486135405))
```

￼￼

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))

返回

[ST_Dump](#), [ST_LocateAlong](#), [ST_LocateBetween](#)

7.19.9 ST_InterpolatePoint

`ST_InterpolatePoint` — 返回沿线性几何体 (M 分量) 的指定位置的插值。

Synopsis

```
float8 ST_InterpolatePoint(geometry linear_geom_with_measure, geometry point);
```

返回

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.

返回

```
SELECT ST_InterpolatePoint('LINESTRING M (0 0 0, 10 0 20)', 'POINT(5 5)');
-----
10
```

返回

[ST_AddMeasure](#), [ST_LocateAlong](#), [ST_LocateBetween](#)

7.19.10 ST_AddMeasure

`ST_AddMeasure` — 沿线性几何体插值。

Synopsis

```
geometry ST_AddMeasure(geometry geom_mline, float8 measure_start, float8 measure_end);
```


2.2.0 繁體中文。



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

SQL

```
-- 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
```

SQL

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);

SQL

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 繁體中文。



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

SQL

```
-- 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_Version](#)

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()**;



Returns the version number of the PROJ library, or against which PostGIS was built.

Availability: 3.5.0



```
SELECT PostGIS_PROJ_Compiled_Version();
 postgis_proj_compiled_version
-----
 9.1.1
(1 row)
```



[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()**;



Returns the version number of the internal Wagyu library, or NULL if Wagyu support is not enabled.



```
SELECT PostGIS_Wagyu_Version();
 postgis_wagyu_version
-----
 0.5.0 (Internal)
(1 row)
```



[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()**;

返回

Returns PostGIS version number and compile-time options.

返回

```
SELECT PostGIS_Version();
               postgis_version
-----
3.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
(1 row)
```

返回

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 目录的路径。如果未设置，则默认为 GDAL_DATA 目录。

返回

GDAL 的 GDAL_DATA 目录的路径 PostgreSQL GUC 设置。postgis.gdal_datapath 的 GDAL 目录的路径。

GDAL 的 GDAL 目录的路径 (hard-coded) 的路径。GDAL 的 GDAL 目录的路径。



Note

PostgreSQL 的 postgresql.conf 文件的路径。如果未设置，则默认为 postgresql.conf 文件的路径。

2.2.0 版本的路径。



Note

GDAL 的 GDAL 目录的路径 GDAL_DATA 目录的路径。

❌❌

postgis.gdal_datapath 安装失败。

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';
SET postgis.gdal_datapath TO default;
```

安装失败。

```
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 使用的 GDAL 驱动程序。GDAL 驱动程序 GDAL_SKIP 驱动程序。

❌❌

PostGIS 使用的 GDAL 驱动程序。GDAL 驱动程序 GDAL_SKIP 驱动程序。PostgreSQL 的 postgresql.conf 文件。驱动程序驱动程序。

PostgreSQL 的 POSTGIS_GDAL_ENABLED_DRIVERS 驱动程序 (pass) 驱动程序 postgis.gdal_enabled_drivers 驱动程序。

GDAL 驱动程序。驱动程序 GDAL 驱动程序。驱动程序。

Note

postgis.gdal_enabled_drivers 驱动程序。驱动程序。



- DISABLE_ALL 驱动程序 GDAL 驱动程序。DISABLE_ALL 驱动程序, postgis.gdal_enabled_drivers 驱动程序。
- ENABLE_ALL 驱动程序 GDAL 驱动程序。
- VSICURL 驱动程序 GDAL 驱动程序 /vsicurl/ 驱动程序。

postgis.gdal_enabled_drivers 驱动程序 DISABLE_ALL 驱动程序, DB 驱动程序, ST_FromGDALRaster(), ST_AsGDALRaster(), ST_AsTIFF(), ST_AsJPEG() 驱动程序 ST_AsPNG() 驱动程序。



Note

驱动程序 PostGIS 驱动程序, postgis.gdal_enabled_drivers 驱动程序 DISABLE_ALL 驱动程序。

**Note**

GDAL_SKIP 选项 GDAL 的 [Configuration Options](#) 选项。

2.2.0 版本。

要

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();
```

要

[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 选项。

要

DB 选项。PostgreSQL 的 `postgresql.conf` 选项。

PostgreSQL 的 `0` 选项 `POSTGIS_ENABLE_OUTDB_RASTERS` 选项 (pass) `postgis.enable_outdb_rasters` 选项。

**Note**

`postgis.enable_outdb_rasters` 选项, GUC `postgis.enable_outdb_rasters` 选项。

**Note**

PostGIS 选项, `postgis.enable_outdb_rasters` 选项。

2.2.0 版本。

配置选项

`postgis.enable_outdb_rasters` 配置选项。

```
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();
```

配置选项

`postgis.gdal_enabled_drivers` `postgis.gdal_vsi_options`

7.22.4 postgis.gdal_vsi_options

`postgis.gdal_vsi_options` — DB 配置选项。

配置选项

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

配置选项

`postgis.enable_outdb_rasters` 配置选项。

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY= ↵
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyy';
```

Set `postgis.gdal_vsi_options` just for the *current transaction* using the `LOCAL` keyword:

```
SET LOCAL postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxx ↵
AWS_SECRET_ACCESS_KEY=yyyyyyyyyyyyyyyyyyyyyyyyyy';
```

配置选项

`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);



Note



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 — 删除几何体的边界框。

Synopsis

geometry **PostGIS_DropBBox**(geometry geomA);

返回

返回几何体的边界框。如果几何体是空值，则返回空值。如果几何体是点，则返回该点的边界框。如果几何体是线，则返回该线的边界框。如果几何体是多边形，则返回该多边形的边界框。如果几何体是几何集合，则返回该集合的边界框。如果几何体是 ST_Intersects 的结果，则返回该结果的边界框。

Note



从 PostGIS 8.3 开始，PostGIS_DropBBox 函数将不再使用缓存的边界框。相反，它将根据几何体的实际形状重新计算边界框。这意味着，对于复杂的几何体，PostGIS_DropBBox 可能会返回一个比缓存的边界框更大的边界框。从 PostGIS 8.3.6 开始，PostGIS_DropBBox 函数将返回一个比缓存的边界框更小的边界框。这意味着，对于复杂的几何体，PostGIS_DropBBox 可能会返回一个比缓存的边界框更小的边界框。从 PostGIS 8.3.6 开始，PostGIS_DropBBox 函数将返回一个比缓存的边界框更小的边界框。这意味着，对于复杂的几何体，PostGIS_DropBBox 可能会返回一个比缓存的边界框更小的边界框。



This method supports Circular Strings and Curves.

返回

```
--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);
```

返回

PostGIS_AddBBox, PostGIS_HasBBox, Box2D

7.23.3 PostGIS_HasBBox

PostGIS_HasBBox — 检查几何体是否具有边界框。

Synopsis

boolean **PostGIS_HasBBox**(geometry geomA);

返回

返回布尔值，表示几何体是否具有边界框。如果具有边界框，则返回 true。如果没有边界框，则返回 false。PostGIS_AddBBox 和 PostGIS_DropBBox 函数用于添加或移除边界框。



This method supports Circular Strings and Curves.

返回

```
SELECT geom
FROM sometable WHERE PostGIS_HasBBox(geom) = false;
```

返回

PostGIS_AddBBox, PostGIS_DropBBox

Chapter 8

SFCGAL Functions Reference

SFCGAL 是一个 2D 和 3D 几何处理库，它封装了 CGAL 库的 C++ 接口 (wrapper)。它提供了许多函数，用于处理几何数据，包括点、线、面、体等。

SFCGAL 的官方网站是 <http://www.sfcgal.org>。您可以在该网站上找到 SFCGAL 的文档和下载链接。SFCGAL 是 PostGIS 的一个扩展，您可以通过安装 PostGIS 来使用 SFCGAL。

8.1 SFCGAL Management Functions

8.1.1 postgis_sfcgal_version

`postgis_sfcgal_version` — 返回 SFCGAL 的版本号。

Synopsis

```
text postgis_sfcgal_version(void);
```

返回

返回 SFCGAL 的版本号。

2.1.0 版本开始支持。



This method needs SFCGAL backend.

返回

`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);

☒☒

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).

8.2.3 CG_IsSolid

CG_IsSolid — 判断几何体是否为实心。返回布尔值。

Synopsis

boolean **CG_IsSolid**(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).

8.2.4 CG_MakeSolid

CG_MakeSolid — 将几何体转换为实心体。如果输入是 Polyhedral 表面，则返回 Polyhedral 表面。如果输入是 Triangles 或 Triangulated Irregular Network Surfaces (TIN)，则返回 TIN。

Synopsis

geometry **CG_MakeSolid**(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).

8.2.5 CG_Orientation

CG_Orientation — 返回 (orientation) 值。

Synopsis

```
integer CG_Orientation(geometry geom);
```

返回

返回值的范围是 -1 到 1。-1 表示逆时针，1 表示顺时针。

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 );
```

返回

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.

返回

```
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)
```

返回

ST_3DArea, ST_Area

8.2.7 CG_3DArea

CG_3DArea — 3 维多面体对象的表面积。返回浮点值 0 或更大。

Synopsis

```
float CG_3DArea(geometry geom1);
```

返回

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).

返回

注意: 返回 KWT 多面体对象的表面积。返回浮点值 0 或更大。返回浮点值, 返回浮点值。

```
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
```

返回

CG_Area, CG_MakeSolid, CG_IsSolid, CG_Area

8.2.8 CG_Volume

CG_Volume — 3 维多面体对象的体积。返回浮点值 (返回浮点值) 0 或更大。

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);
```


ST_ForceLHR is deprecated as of 3.5.0. Use **CG_ForceLHR** instead.



8.2.10 ST_IsPlanar



ST_IsPlanar is deprecated as of 3.5.0. Use **CG_IsPlanar** instead.



8.2.11 ST_IsSolid

```
boolean ST_IsSolid(geometry geom1);
```


☒☒



Warning
ST_IsSolid is deprecated as of 3.5.0. Use **CG_IsSolid** instead.

2.2.0 简体中文.

- ✔ 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 — 简体中文. 简体中文. 简体中文, 简体中文 简体中文 TIN 简体中文.

Synopsis

geometry **ST_MakeSolid**(geometry geom1);

☒☒



Warning
ST_MakeSolid is deprecated as of 3.5.0. Use **CG_MakeSolid** instead.

2.2.0 简体中文.

- ✔ 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 — 简体中文 (orientation) 简体中文.

Synopsis

integer **ST_Orientation**(geometry geom);



Warning

ST_Orientation is deprecated as of 3.5.0. Use **CG_Orientation** instead.

□□□□□□□□□□□□□□. □□□□□□□□□□□□□□ -1 □, □□□□□□□□ 1 □□□□□□□□.

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 字节的浮点型值。 0 表示空值。

Synopsis

```
floatST_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).

[illegible]

```
SELECT ST_3DArea(geom) As cube_surface_area,
       ST_3DArea(ST_MakeSolid(geom)) As solid_surface_area
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);

cube_surface_area | solid_surface_area
-----+-----
6 | 0
```

☐☐

[ST_Area](#), [ST_MakeSolid](#), [ST_IsSolid](#), [ST_Area](#)

8.2.15 ST_Volume

ST_Volume — 3 维几何体的体积。 几何体 (多面体) 的 0 维几何体。

Synopsis

float **ST_Volume**(geometry geom1);

☐☐



Warning

ST_Volume is deprecated as of 3.5.0. Use **CG_Volume** instead.

2.2.0 简体中文。

- ✓ This method needs SFCGAL backend.
- ✓ This function supports 3d and will not drop the z-index.
- ✓ This function supports Polyhedral surfaces.
- ✓ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✓ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 9.1 (same as ST_3DVolume)

☐☐

WKT 多面体, 多面体。 多面体, **ST_MakeSolid** 多面体。 多面体。

```

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);



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).



```
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)
```



ST_3DDifference, ST_Difference

8.3.5 ST_3DDifference

ST_3DDifference — 3 .

Synopsis

geometry **ST_3DDifference**(geometry geom1, geometry geom2);

**Warning**

ST_3DDifference is deprecated as of 3.5.0. Use **CG_3DDifference** instead.

geom2 返回 geom1 的副本。

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 维几何体差。

Synopsis

geometry **CG_3DDifference**(geometry geom1, geometry geom2);

返回

geom2 返回 geom1 的副本。

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** 返回 3 维几何体的 X3Dom HTML 格式。HTML 格式。


```
SELECT CG_Extrude(ST_Buffer(
  ST_GeomFromText('POINT(100 90)'),
  50, '
  quad_segs=1'),0,0,30) AS geom2;
```

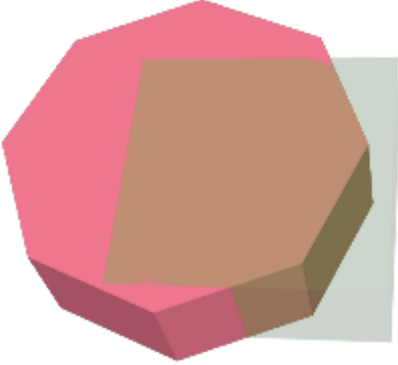
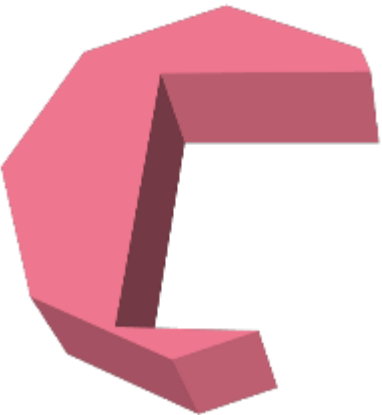


图 3 显示了 `geom2` 的 3D 渲染结果。

```
SELECT CG_3DDifference(geom1,geom2)
  50, '
  quad_segs=2'),0,0,30) AS geom1,
  50, '
  quad_segs=2'),0,0,30) AS geom1,
  CG_Extrude(
  ST_Buffer(ST_GeomFromText('POINT(80 80)'),
  50, '
  quad_segs=1'),0,0,30) AS geom2 ) As t;
```



`geom2` 的 3D 渲染结果。

`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

[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);

See

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

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

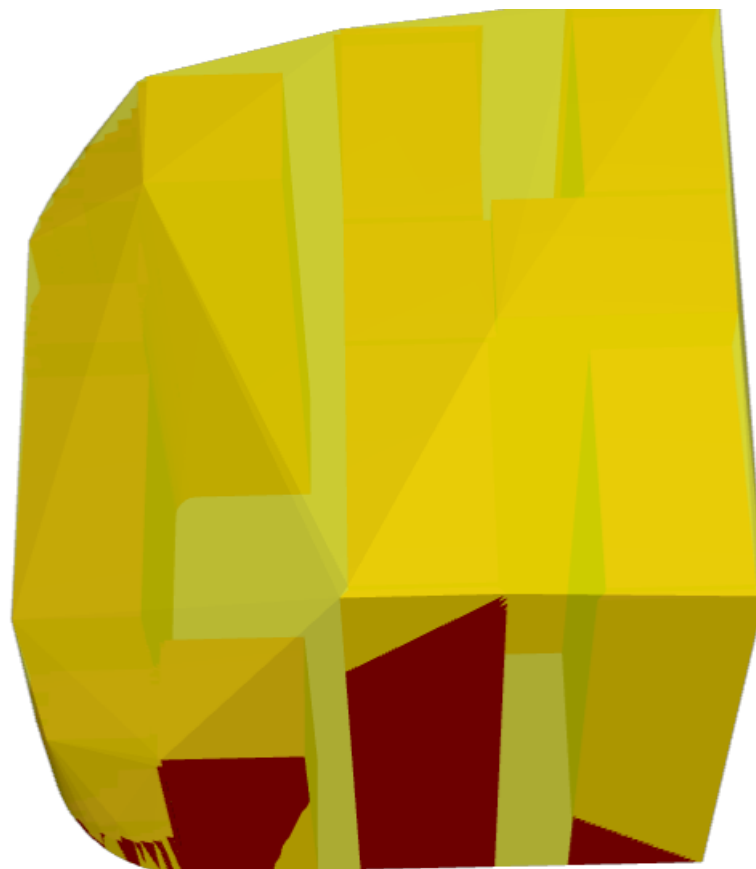
- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

图例

```
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

图例

[ST_Letters](#), [ST_AsX3D](#)

8.3.11 ST_3DIntersection

ST_3DIntersection — 3 个 3D 几何体的交集。

```
geometry ST_3DIntersection(geometry geom1, geometry geom2);
```



ST_3DIntersection is deprecated as of 3.5.0. Use **CG_3DIntersection** instead.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.

- ✔ This method needs SFCGAL backend.
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

CG_3DIntersection — 3 .

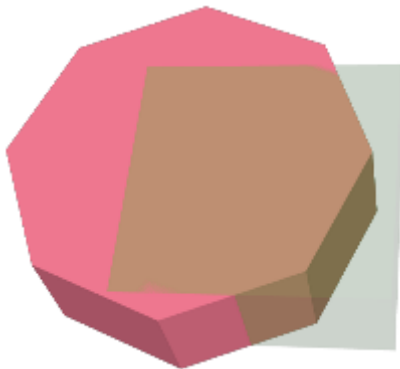
```
geometry CG_3DIntersection(geometry geom1, geometry geom2);
```

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).

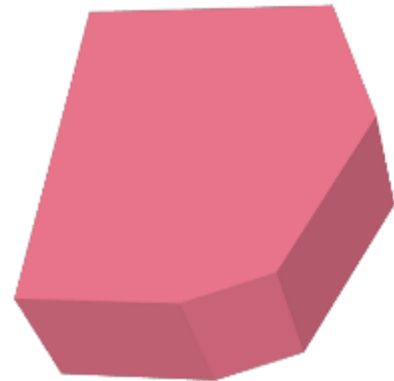
PostGIS **ST_AsX3D** 3D DOM HTML X3Dom HTML HTML.

```
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 的 3D 交集。 geom2 的 3D 交集。

```
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('POINT(80 80)'),
    50, '
quad_segs=1'),0,0,30) AS geom2 ) As t;
```



geom1 的 geom2 的 3D 交集

3 的 3D 交集

```
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)
```

3D 交集 (3D 交集) 的 3D 交集

```
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)))
```

3D 交集 (3D 交集) 的 3D 交集 (ST_Dimension 3 的 3D 交集)

```
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 XXXXXXXXXX.

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);



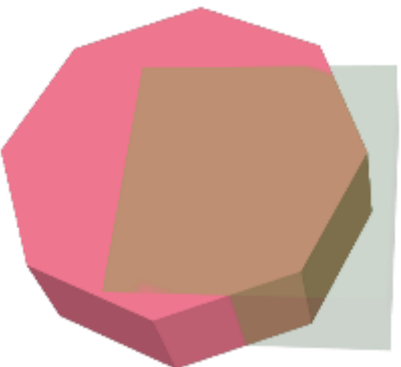
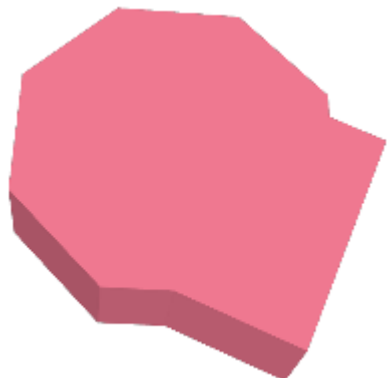
Availability: 3.5.0

- ✔ 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** 3D **X3Dom** HTML **3D** **HTML**.

<pre>SELECT CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 'quad_segs=1'),0,0,30) AS geom2;</pre>  3 3D visualization of the result of the SQL query. The result shows a 3D visualization of the result of the SQL query. The result shows a 3D visualization of the result of the SQL query.	<pre>SELECT CG_3DUnion(geom1,geom2)</pre>  3D visualization of the result of the SQL query. The result shows a 3D visualization of the result of the SQL query.

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.

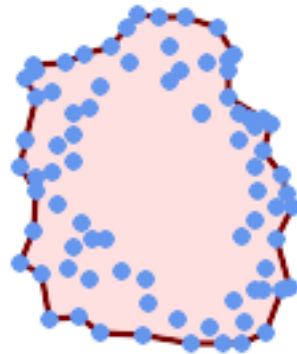
This function effectively computes a concave hull of a geometry in a similar way to **ST_ConcaveHull**, but uses CGAL and a different algorithm.

Availability: 3.5.0 - requires SFCGAL >= 1.4.1.



This method needs SFCGAL backend.

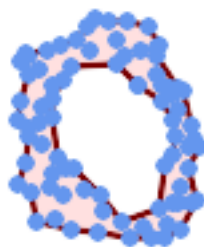
☒☒



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));
```

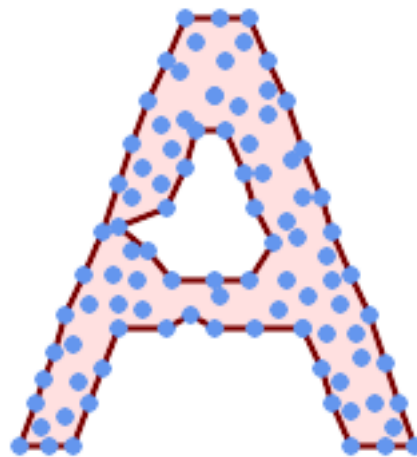
```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,
37 23,30 22,28 33,23 36,26 44,27 54,23 60,24 67,27 77,
24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,
64 97,72 95,76 88,75 84,83 72,85 71,88 58,89 53))
```



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.

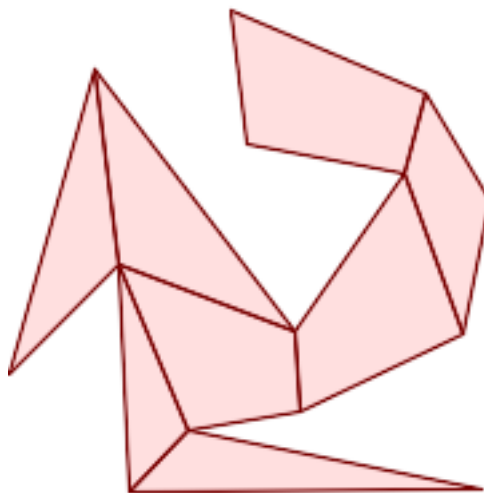
Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.

￼￼



Approximal Convex Partition (same example As [CG_YMonotonePartition](#), [CG_GreeneApproxConvexPartition](#) and [CG_OptimalConvexPartition](#))

```
SELECT ST_AsText(CG_ApproxConvexPartition('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((107 61,32 159,41 86,107 61)),POLYGON((45 1,177 2,67 24,45 1)),POLYGON((41 86,45 1,67 24,41 86)),POLYGON((107 61,41 86,67 24,109 31,107 61)),POLYGON((148 120,107 61,109 31,170 60,148 120)),POLYGON((156 150,148 120,170 60,180 110,156 150)))
```

￼￼

[CG_YMonotonePartition](#), [CG_GreeneApproxConvexPartition](#), [CG_OptimalConvexPartition](#)

8.3.19 ST_ApproximateMedialAxis

ST_ApproximateMedialAxis — ￼￼￼￼￼￼￼￼￼￼￼￼.

Synopsis

geometry **ST_ApproximateMedialAxis**(geometry geom);

￼￼



Warning

ST_ApproximateMedialAxis is deprecated as of 3.5.0. Use [CG_ApproximateMedialAxis](#) instead.

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);



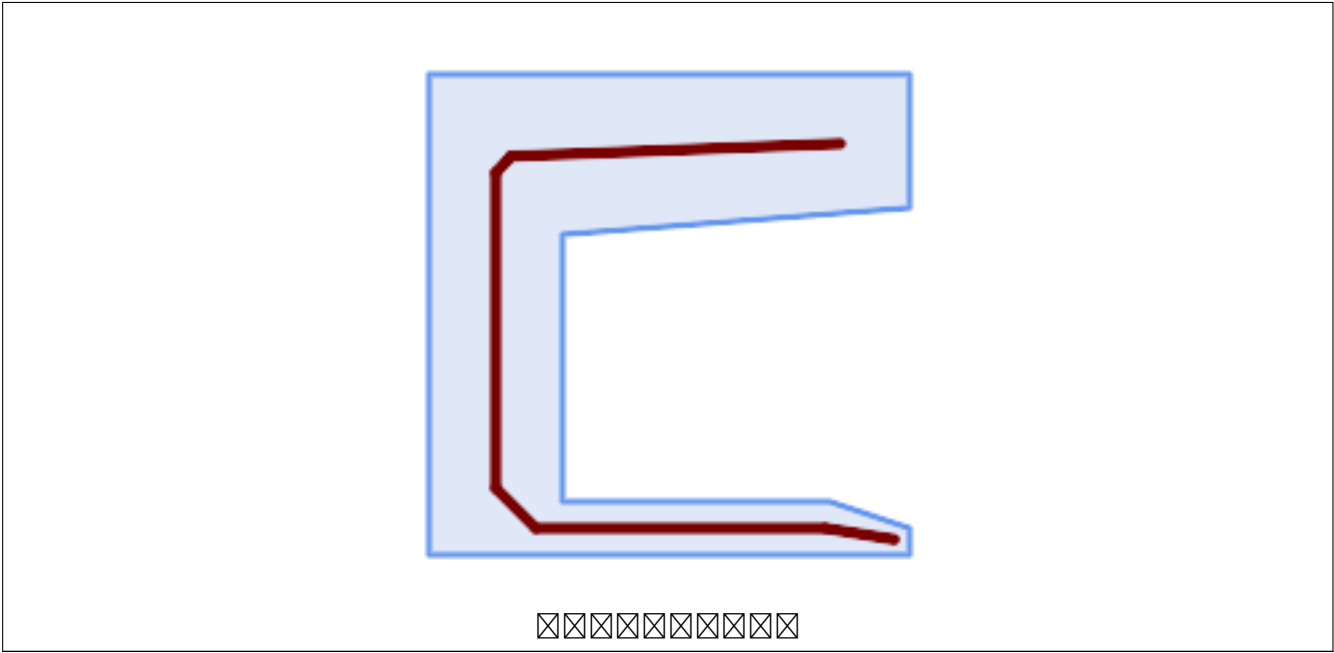
Return an approximate medial axis for the areal input based on its straight skeleton. Uses an SFCGAL specific API when built against a capable version (1.2.0+). Otherwise the function is just a wrapper around `CG_StraightSkeleton` (slower case).

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 ))'));
```



图例

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);

图例



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 简体中文.



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

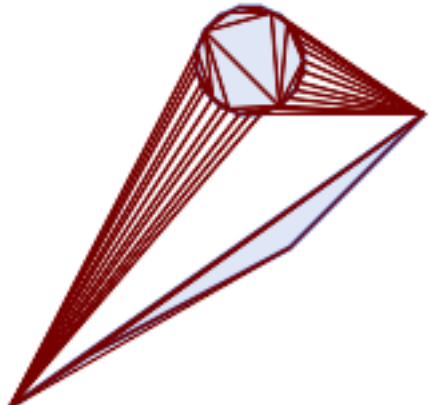
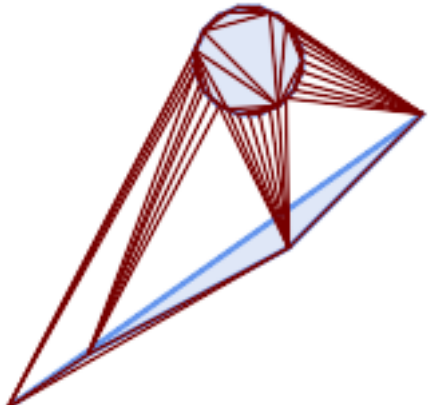
geometry **CG_ConstrainedDelaunayTriangles**(geometry g1);

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.

 <i>CG_ConstrainedDelaunayTriangles of 2 polygons</i> <pre>select CG_ConstrainedDelaunayTriangles(ST_Union(POLYGON((175 150, 20 40, 50 60, 125 100, 175 150)), ST_Buffer('POINT(110 170)::geometry', 20)));</pre>	 <i>ST_DelaunayTriangles of 2 polygons. Triangle edges cross polygon boundaries.</i> <pre>select ST_DelaunayTriangles(ST_Union(POLYGON((175 150, 20 40, 50 60, 125 100, 175 150)), ST_Buffer('POINT(110 170)::geometry', 20)));</pre>

ST DelaunayTriangles, ST TriangulatePolygon, CG Tesselate, ST ConcaveHull, ST Dump

8.3.23 ST Extrude

ST Extrude — □□□□□□□□□□□□.

Synopsis

```
geometry ST_Extrude(geometry geom, float x, float y, float z);
```



Warning

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

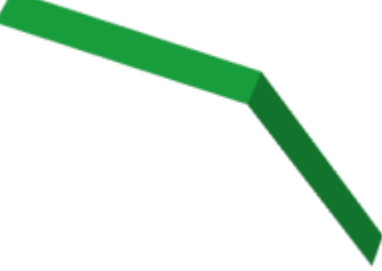
```
geometry CG Extrude(geometry geom, float x, float y, float z);
```


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).

图

PostGIS **ST_AsX3D** 返回 3 维几何体的 X3Dom HTML 或 SVG 格式。HTML 格式支持 3D 渲染。

<pre>SELECT ST_Buffer(ST_GeomFromText('POINT (100 90)'), 50, 'quad_segs=2',0,0,30);</pre>  图例	<pre>CG_Extrude(ST_Buffer(ST_GeomFromText('POINT (100 90)'), 50, 'quad_segs=2',0,0,30);</pre>  Z 轴高度 30 的 3D 几何体 <i>Z(PolyhedralSurfaceZ)</i> 格式。
<pre>SELECT ST_GeomFromText('LINESTRING(50 50, 100 90, 95 150)')</pre>  图例	<pre>SELECT CG_Extrude(ST_GeomFromText('LINESTRING(50 50, 100 90, 95 150)')</pre>  Z 轴高度 30 的 3D 几何体 <i>Z(PolyhedralSurfaceZ)</i> 格式。

☒☒

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 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))
,((4 5 2,5 5 2,4 4 2,4 5 2)),((2 1 2,5 0 2,0 0 2,2 1 2)),((5 5 2,5 0 2,4 4 2,5 5 2)),((2
1 2,0 0 2,1 1 2,2 1 2)),((1 2 2,1 1 2,0 0 2,1 2 2)),((0 4 2,2 2 2,1 2 2,0 4 2)),((0 4
2,1 2 2,0 0 2,0 4 2)),((4 4 2,5 0 2,2 2 2,4 4 2)),((4 4 2,2 2 2,0 4 2,4 4 2)),((2 2 2,5
0 2,2 1 2,2 2 2)),((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
3.5 2.5 3.5))
```

```

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.

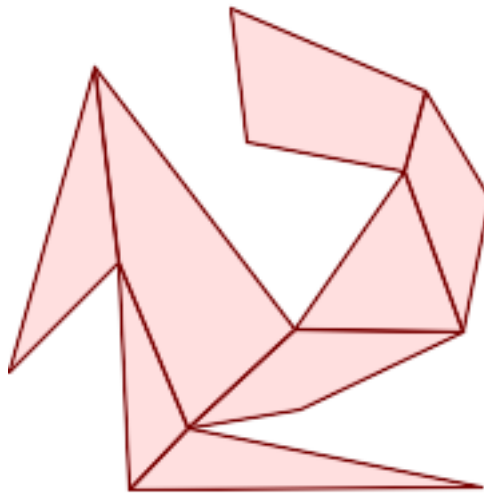
Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

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`, 参数是 `geometry` 2 个 `geometry` 类型。

返回 A 与 B 的 Minkowski 和。A 与 B 的 Minkowski 和是 A 与 B 的并集的凸包。Minkowski 和 (motion planning) 与 CAD(computer-aided design) 密切相关。Wikipedia Minkowski addition 了解更多。

返回类型是 `geometry` 2 个 `geometry` (点, 线, 面) 的并集。返回 3 个 `geometry` 类型, Z 坐标为 0 的 `geometry` 2 个 `geometry` 类型。Minkowski 和是 2 个 `geometry` 类型的并集。

返回类型 `CGAL 2D Minkowski sum` 了解更多。

2.1.0 版本支持。

 This method needs SFCGAL backend.

8.3.28 CG_MinkowskiSum

`CG_MinkowskiSum` — 计算 Minkowski 和。

Synopsis

`geometry CG_MinkowskiSum(geometry geom1, geometry geom2);`

返回

返回类型是 `geometry`, 参数是 `geometry` 2 个 `geometry` 类型。

返回 A 与 B 的 Minkowski 和。A 与 B 的 Minkowski 和是 A 与 B 的并集的凸包。Minkowski 和 (motion planning) 与 CAD(computer-aided design) 密切相关。Wikipedia Minkowski addition 了解更多。

返回类型是 `geometry` 2 个 `geometry` (点, 线, 面) 的并集。返回 3 个 `geometry` 类型, Z 坐标为 0 的 `geometry` 2 个 `geometry` 类型。Minkowski 和是 2 个 `geometry` 类型的并集。

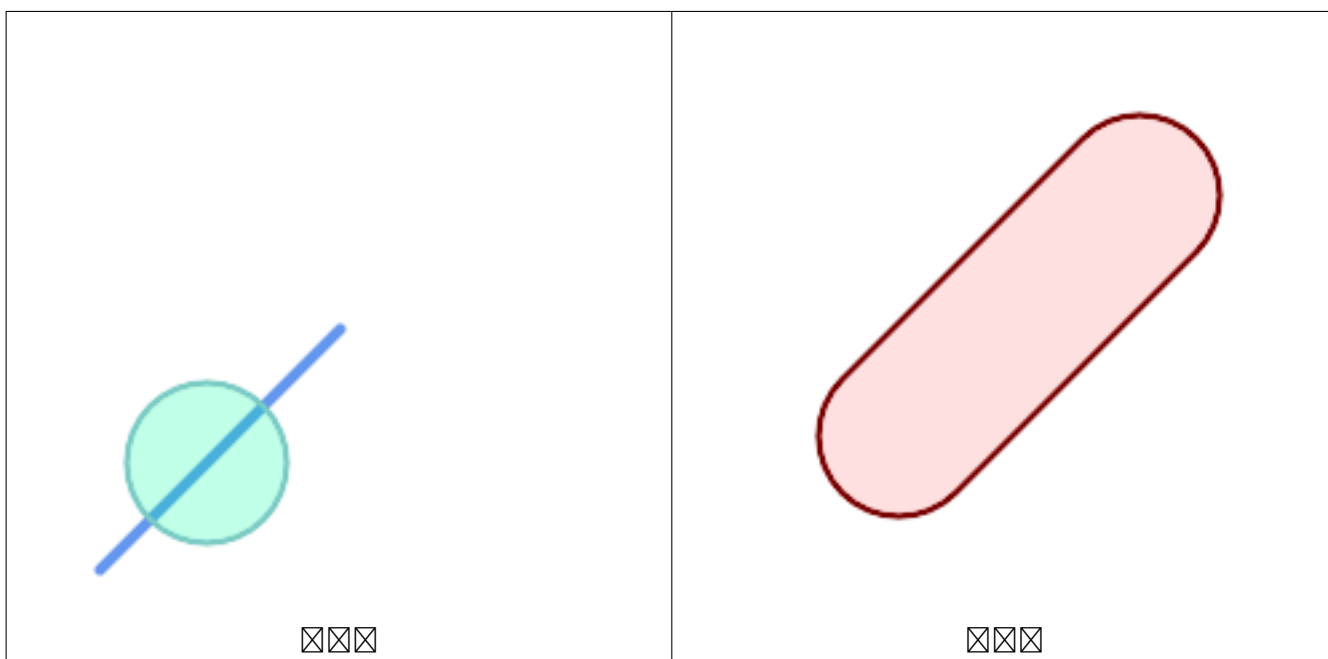
返回类型 `CGAL 2D Minkowski sum` 了解更多。

Availability: 3.5.0

 This method needs SFCGAL backend.

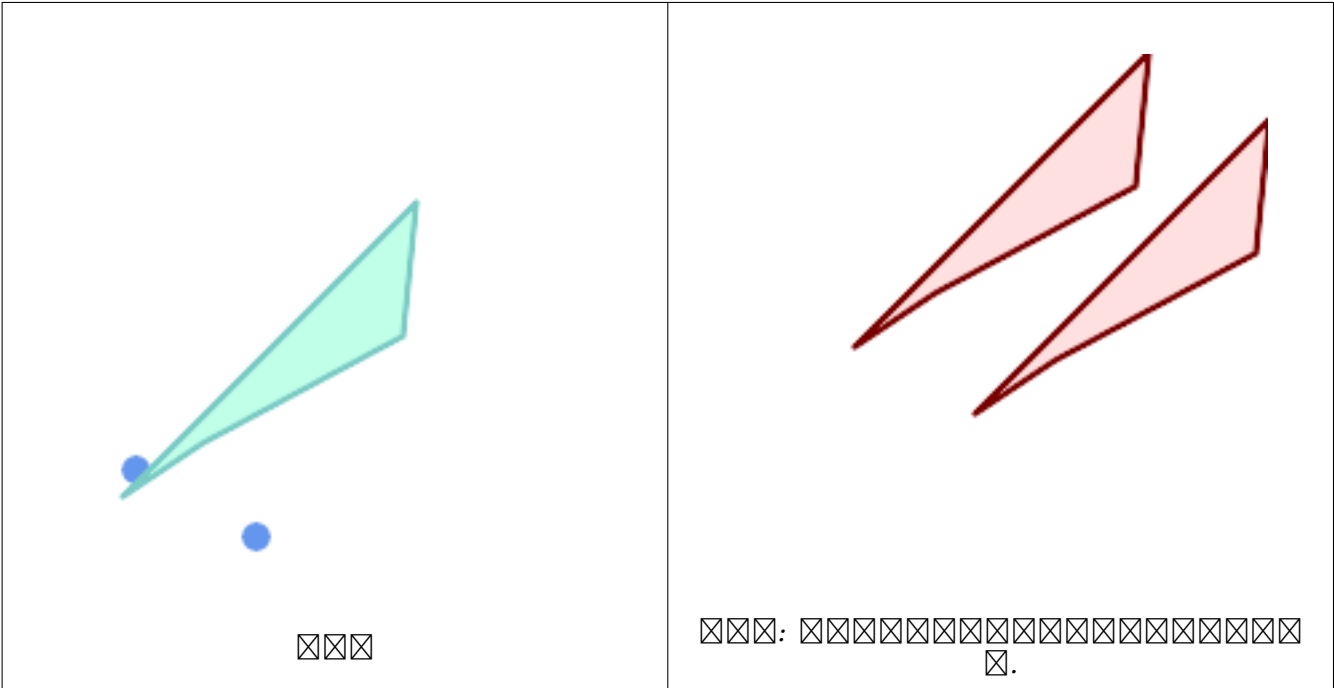
返回

返回类型是 `geometry` 2 个 `geometry` 类型。



```
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 30,65.8527096604839 ↵
30.5764415879031,71.4805029709527 32.2836140246614,76.6671069905881 ↵
35.0559116309237,81.2132034355964 38.7867965644036,171.213203435596 ↵
128.786796564404,174.944088369076 133.332893009412,177.716385975339 ↵
138.519497029047,179.423558412097 144.147290339516,180 150,179.423558412097 ↵
155.852709660484,177.716385975339 161.480502970953,174.944088369076 ↵
166.667106990588,171.213203435596 171.213203435596,166.667106990588 ↵
174.944088369076,
161.480502970953 177.716385975339,155.852709660484 179.423558412097,150 ↵
180,144.147290339516 179.423558412097,138.519497029047 ↵
177.716385975339,133.332893009412 174.944088369076,128.786796564403 ↵
171.213203435596,38.7867965644035 81.2132034355963,35.0559116309236 ↵
76.667106990588,32.2836140246614 71.4805029709526,30.5764415879031 ↵
65.8527096604838,30 59.9999999999999)))
```

```
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);

注意



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 α 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 α 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

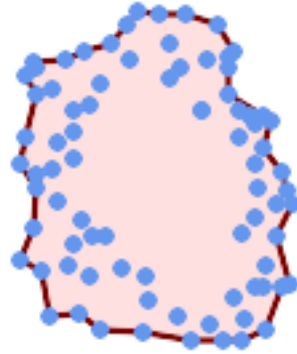
The result will not contain holes unless the optional `allow_holes` argument is specified as true.

Availability: 3.5.0 - requires SFCGAL $\geq$ 1.4.1.



This method needs SFCGAL backend.

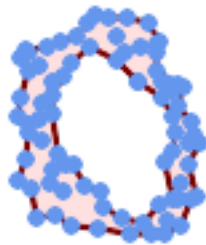
☒☒



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**

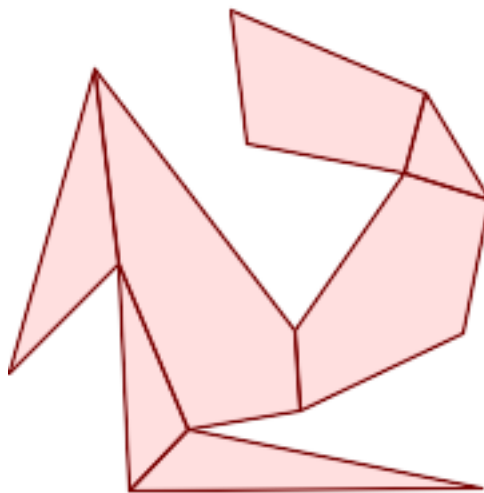
A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. `CG_OptimalConvexPartition` produces a partition that is optimal in the number of pieces.

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);

国国

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).

国国

```
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));
```

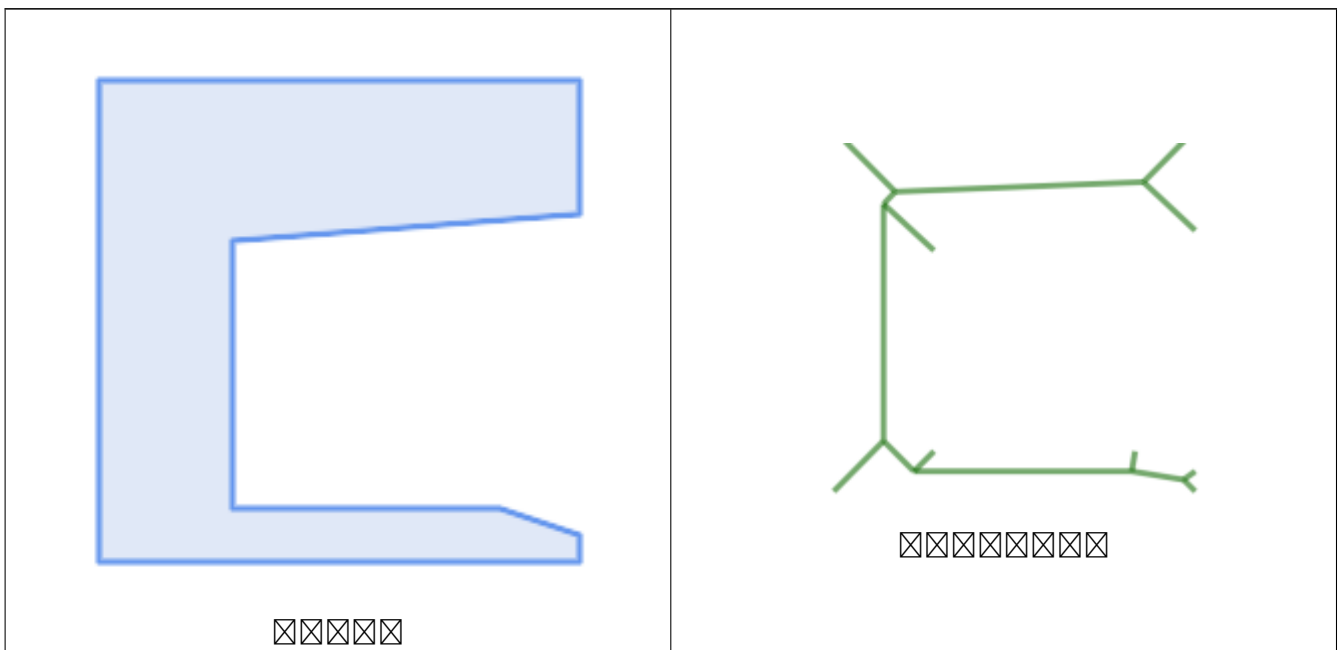
Note that valid inputs with rings that touch at a single point will raise an error.

```
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.



☐☐

[CG_StraightSkeletonPartition](#), [CG_ExtrudeStraightSkeleton](#)

8.3.33 ST_StraightSkeleton

ST_StraightSkeleton — ☐☐☐☐☐☐☐☐☐ (straight skeleton) ☐☐☐☐☐☐.

Synopsis

geometry **ST_StraightSkeleton**(geometry geom);

☐☐



Warning

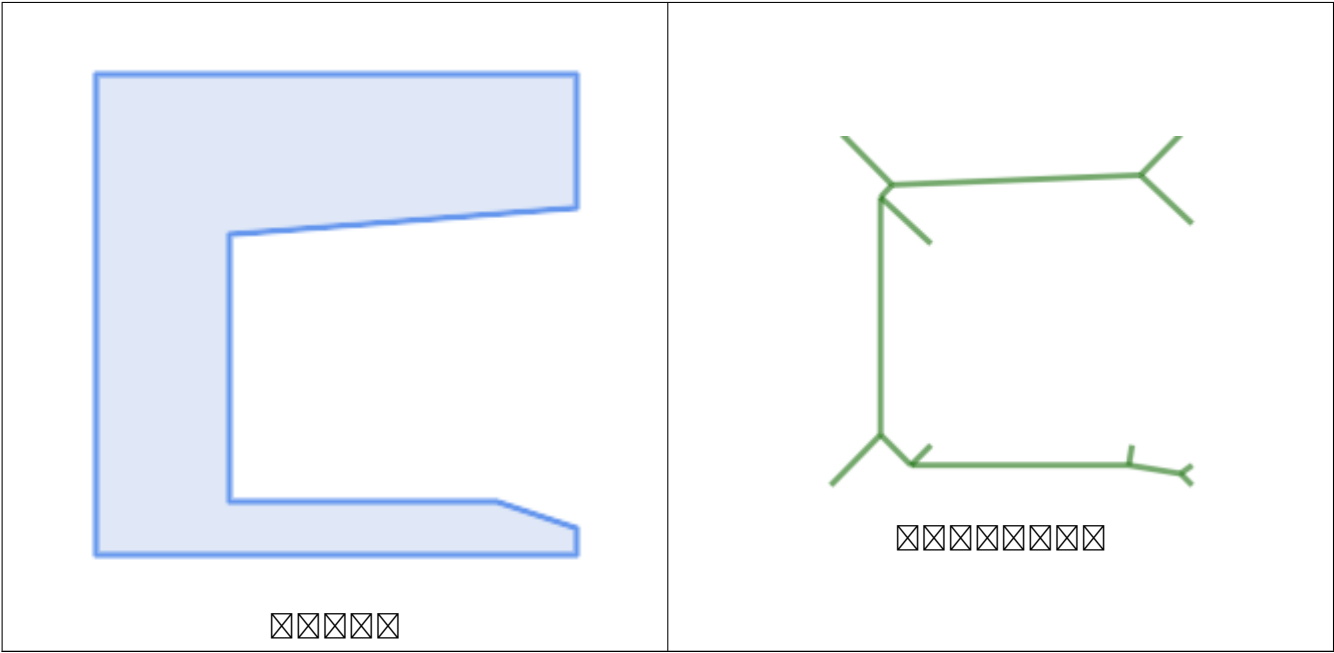
[ST_StraightSkeleton](#) is deprecated as of 3.5.0. Use [CG_StraightSkeleton](#) instead.

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).

☐☐

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



图例

`CG_ExtrudeStraightSkeleton`

8.3.34 ST_Tessellate

ST_Tessellate — 将多边形分解为三角形 (tessellation) 返回 TIN 或 TIN 集合。

Synopsis

geometry **ST_Tessellate**(geometry geom);

图例



Warning
`ST_Tessellate` is deprecated as of 3.5.0. Use `CG_Tessellate` instead.

[图例] 将多边形分解为三角形 (tessellation) 返回 TIN 或 TIN 集合。



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 简体中文。

- ✔ 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 — 简体中文 (tessellation) 简体中文 TIN 简体中文 简体中文。

Synopsis

geometry **CG_Tessellate**(geometry geom);

简体中文

[简体中文] 简体中文 (简体中文) 简体中文 TIN 简体中文。



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).

简体中文

```
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
```



ST_GeomFromText

```
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

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



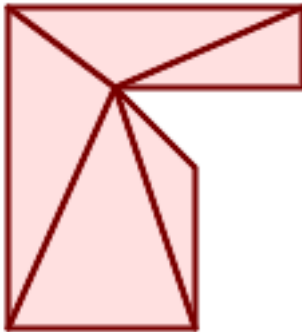
CG_Tessellate

```
SELECT 'POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160, 120 160, 120 190, 10 190 ))'
```



简体中文

```
SELECT
    CG_Tessellate('
POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160,
;
ST_AsText 中文:
geometry;
TIN((80 130,50 160,80 70,80 130)),((50 160,10 190,10 70,50 160)),
((80 70,50 160,10 70,80 70)),((120 160,120 190,50 160,120 160)),
((120 190,10 190,50 160,120 190)))
```



简体中文

中文

[CG_ConstrainedDelaunayTriangles](#), [ST_DelaunayTriangles](#), [ST_TriangulatePolygon](#)

8.3.36 CG_Triangulate

CG_Triangulate — Triangulates a polygonal geometry

Synopsis

geometry **CG_Triangulate**(geometry geom);

中文

Triangulates a polygonal geometry.

Performed by the SFCGAL module

**Note**

NOTE: this function returns a geometry representing the triangulated result.

Availability: 3.5.0



This method needs SFCGAL backend.

SQL

```
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)
```

SQL

[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);
```

SQL

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.



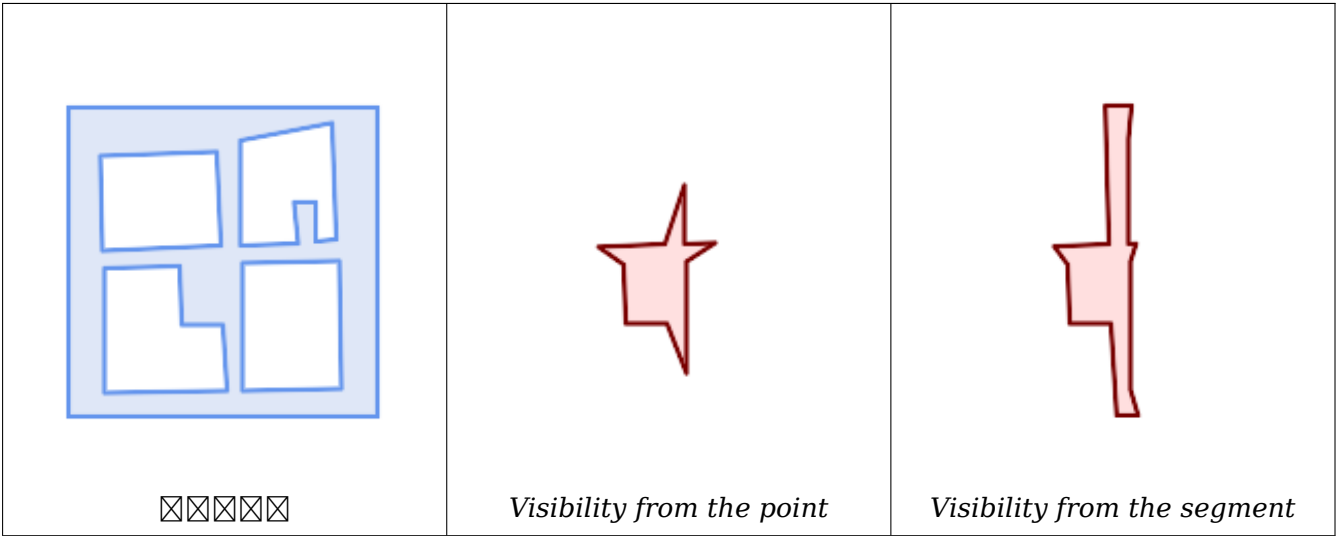
This function supports Polyhedral surfaces.



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.

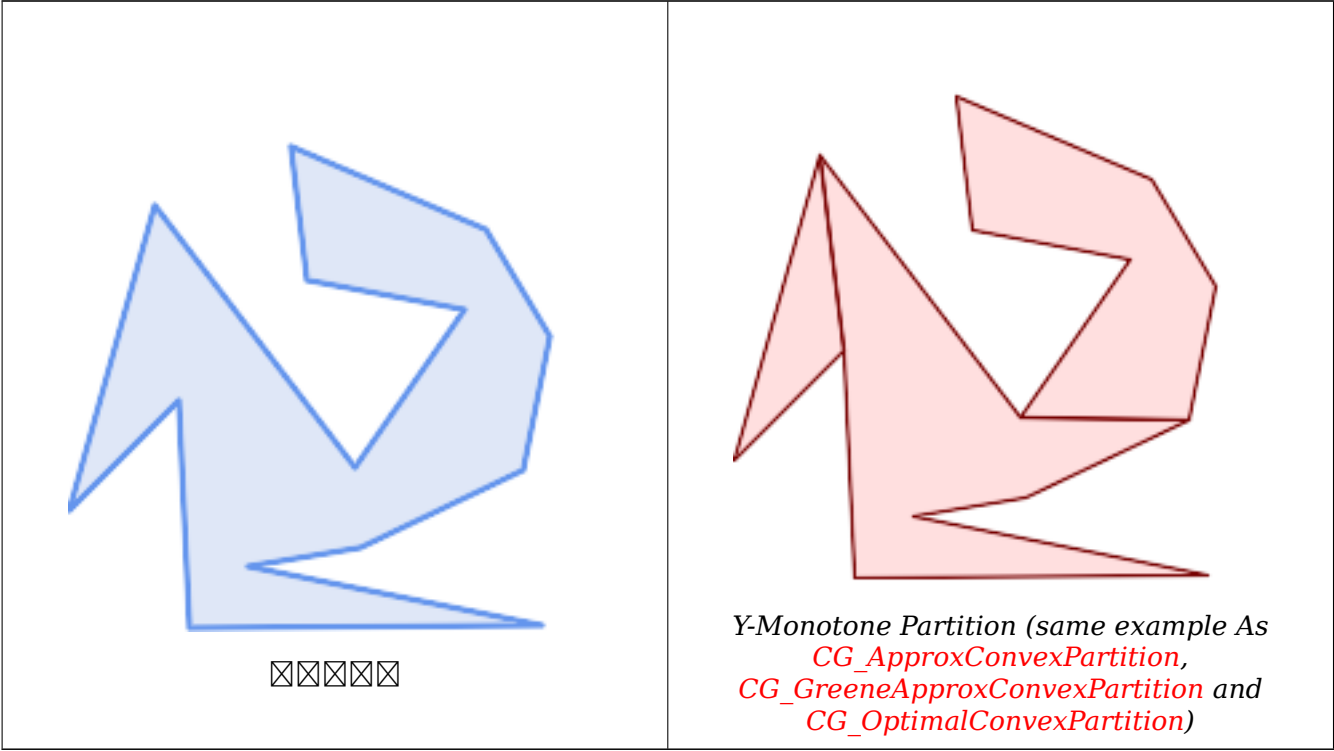
**Note**

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);

国国

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.

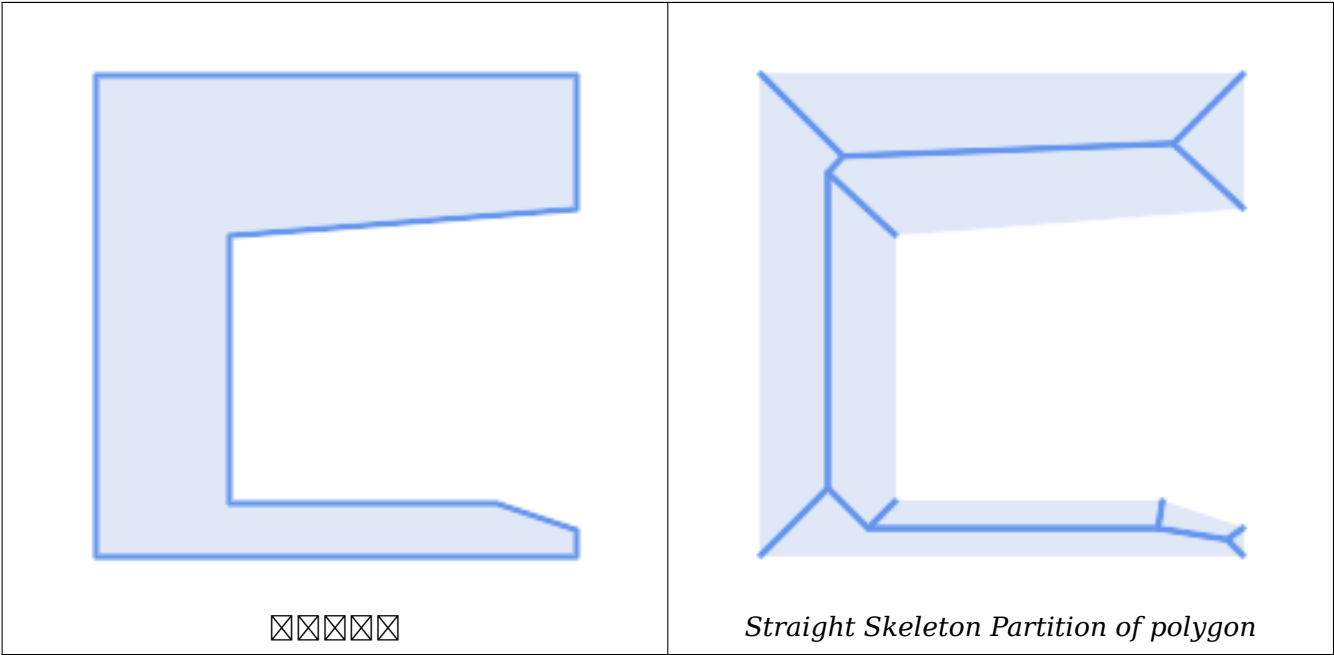
Availability: 3.6.0 - requires SFCGAL >= 2.0.0.

✔ This method needs SFCGAL backend.

国国

```
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 );
```



国国

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



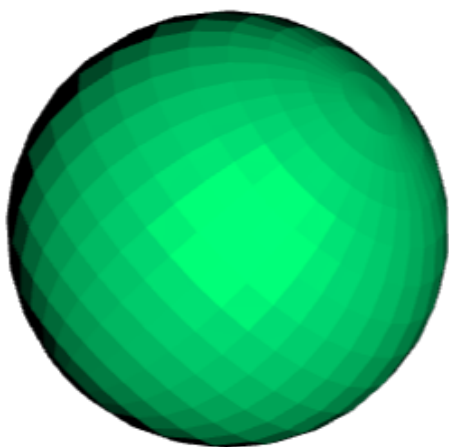
This method needs SFCGAL backend.



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



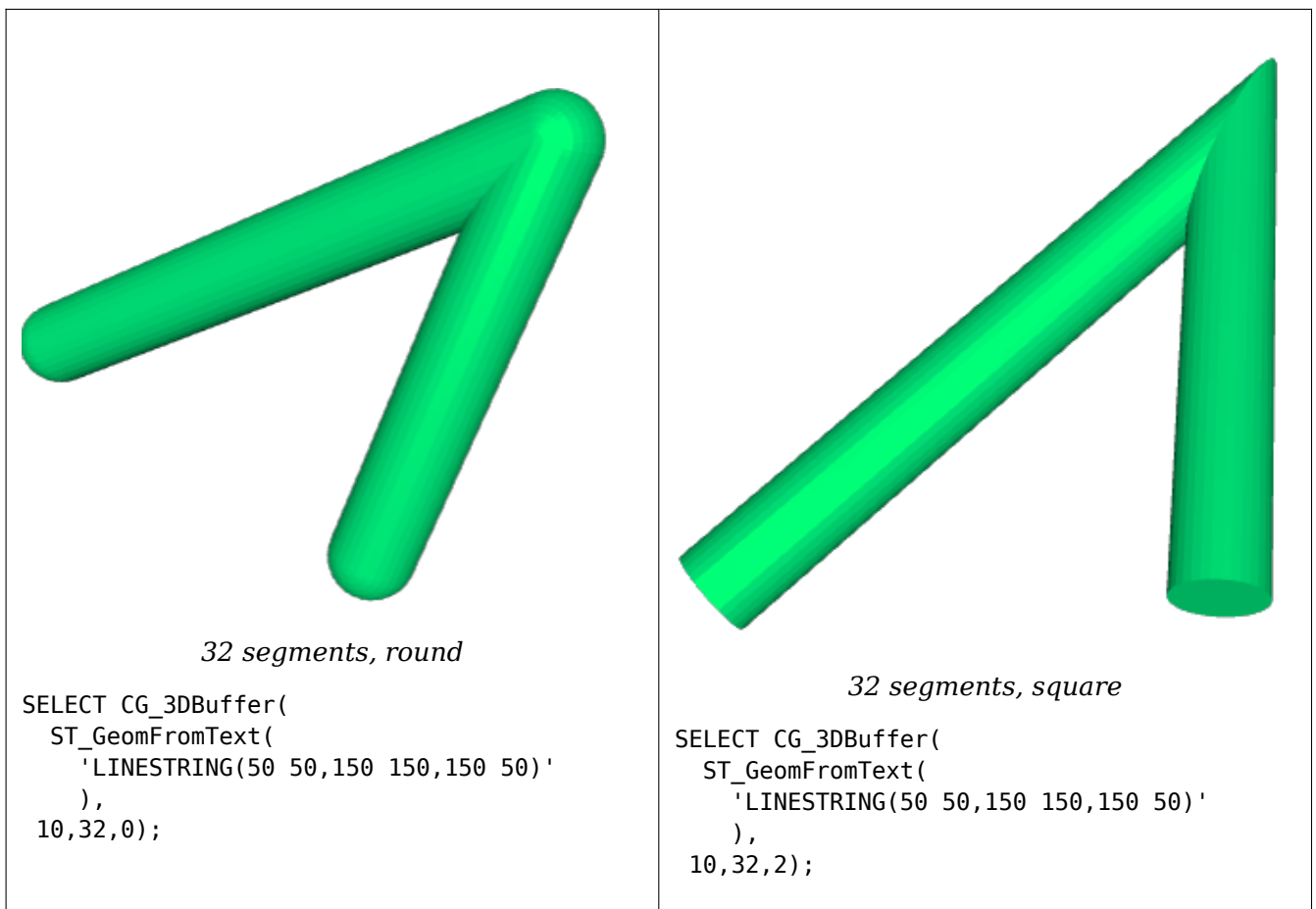
segments=32 (rounded buffer)

```
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.

￼￼

```
SELECT ST_AsText(CG_3DScale('POINT(1 1 1)', 2, 3, 4));  
-- Result: POINT(2 3 4)
```

￼￼

[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);

￼￼

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.

￼￼

```
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)
```

￼￼

[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));

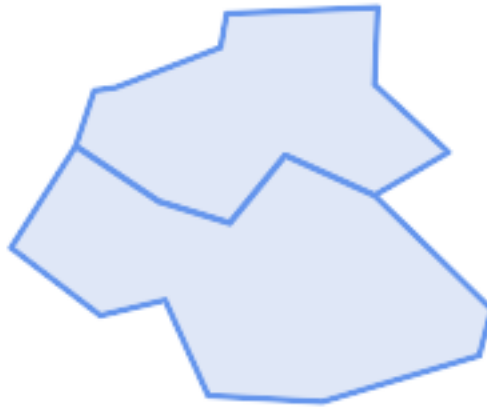
-- Simplify a geometry with both Z and M values
SELECT ST_AsText(CG_Simplify(ST_GeomFromText('LINESTRING ZM(0 0 0 1, 0 1 1 2, 0.1 1 1 3, 0.2 1 1 4, 0.3 1 1 5, 1 1 2 6)'), 0.2));
```



```
-- 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));

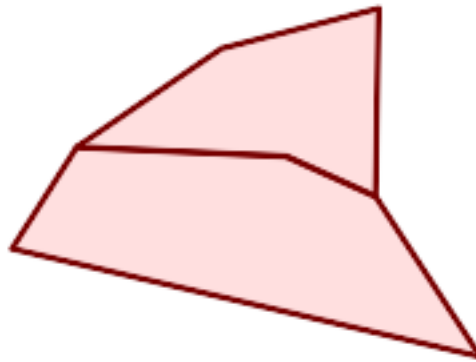
-- Simplify two geometry together preserving Z and M values, with topology
SELECT ST_AsText(CG_Simplify('GEOMETRYCOLLECTION ZM(LINESTRING ZM(-1 -1 3 4, 0 0 10 100, 1 1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150), POLYGON ZM((0 0 10 100, 1 1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -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;
```



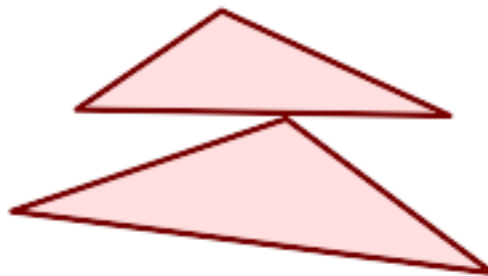
Originals geometries

```
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



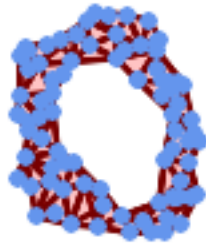
This method needs SFCGAL backend.



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

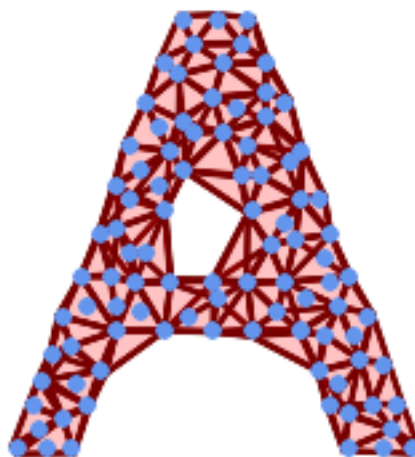
☒☒

```
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);
```

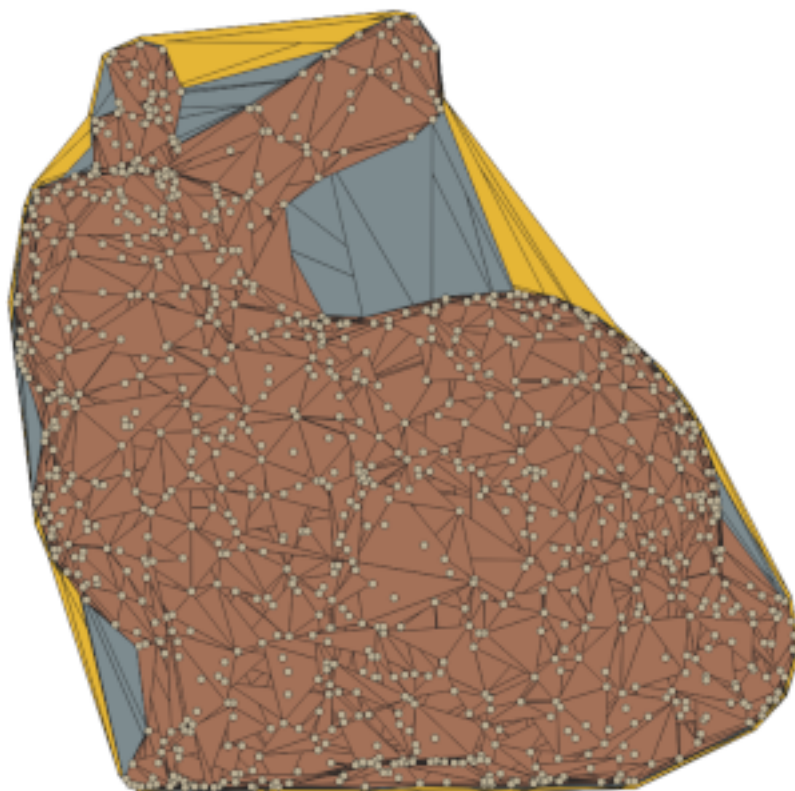


Alpha wrapping of a MultiPoint (same example As [CG_OptimalAlphaShape](#))

```
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.

简介

TopoGeometry 是 PostGIS 3.6.0 引入的一个新数据类型，用于表示拓扑定义的几何。TopoGeometry 由 topology_id, layer_id, id, type 四个属性组成。

1. **topology_id** (integer): 表示 TopoGeometry 所属的 topology 的 ID。SRID 由 topology.topology SRID 决定。
2. **layer_id** (integer): TopoGeometry 所属的 layer 的 ID。topology_id 和 layer_id 共同唯一标识 topology.layers 表中的一条记录 (unique reference)。
3. **id** (integer): 表示 TopoGeometry 在 layer 中的唯一标识。
4. **type** (integer): 表示 TopoGeometry 的几何类型。1: [点] 点, 2: [线] 线, 3: [面] 面, 4: [体] 体。

结构

TopoGeometry 的存储格式如下：

topology_id	layer_id
id	type

创建

CreateTopoGeom

9.1.3 validate_topology_returntype

validate_topology_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.

简介

validate_topology_returntype 是 PostGIS 3.6.0 引入的一个新数据类型，用于表示拓扑验证的结果。ValidateTopology 函数返回 validate_topology_returntype 类型的结果，包含错误消息和位置 ID。

1. **error** (varchar): 错误消息。
可能的错误消息 (descriptor) 包括：coincident nodes(重合节点), edge crosses node(边穿过节点), edge not simple(边不简单), edge end node geometry mismatch(边端点几何不匹配), edge start node geometry mismatch(边起始点几何不匹配), face overlaps face(面重叠), face within face(面在面内)。
2. **id1** (integer): 错误涉及的几何对象的 ID (边/面/节点)。
3. **id2** (integer): 错误涉及的另一个几何对象的 ID。



ValidateTopology

9.2 ☐ ☐ ☐ ☐ ☐

9.2.1 TopoElement

TopoElement — TopoGeometry 2



TopoGeometry 1 2 .

TopoGeometry, TopoGeometry (topological primitive) TopoGeometry
 TopoGeometry (1: node, 2: edge, 3: face) TopoGeometry.
 TopoGeometry TopoGeometry.



Note!

Note

TopoGeometry 对象, TopoGeometry 对象, TopoGeometry 对象
topology.layer 对象。



```
SELECT te[1] AS id, te[2] AS type FROM
( SELECT ARRAY[1,2]::topology.topoelement AS te ) f;
```

id	type
1	2

```
SELECT ARRAY[1,2]::topology.topoelement;
      te
-----
 {1,2}
```

```
--Example of what happens when you try to case a 3 element array to topoelement
-- NOTE: topoement has to be a 2 element array so fails dimension check
SELECT ARRAY[1,2,3]::topology.topoelement;
ERROR:  value for domain topology.topoelement violates check constraint "dimensions"
```



GetTopoGeomElements, TopoElementArray, TopoGeometry, TopoGeom addElement, TopoGeom remEleme

9.2.2 TopoElementArray

TopoElementArray — An array of TopoElement objects.

例

1 创建 TopoElement 表, 创建 TopoElement 表。

例

```
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"
```

例

[TopoElement](#), [GetTopoGeomElementArray](#), [TopoElementArray_Agg](#)

9.3 创建 TopoGeometry 表

9.3.1 AddTopoGeometryColumn

AddTopoGeometryColumn — 创建 TopoGeometry 表, topology.layer 表, layer_id 表。

Synopsis

```
integer AddTopoGeometryColumn(varchar topology_name, varchar schema_name, varchar table_name,
varchar column_name, varchar feature_type);
integer AddTopoGeometryColumn(varchar topology_name, varchar schema_name, varchar table_name,
varchar column_name, varchar feature_type, integer child_layer);
```

例

创建 TopoGeometry 表。 TopoGeometry 表。 AddTopoGeometryColumn() 函数。

topology.layer 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');
```

❏

AddTopoGeometryColumn, RenameTopology

9.3.3 DropTopology

DropTopology — 删除拓扑: 删除 topology.topology 表中的记录, 并删除 geometry_columns 表中的记录。

Synopsis

```
integer DropTopology(varchar topology_schema_name);
```

❏

删除 topology.topology 表中的记录, 并删除 geometry_columns 表中的记录。删除记录时, 将 topology_schema_name 替换为要删除的拓扑名称。删除记录时, 将 topology_schema_name 替换为要删除的拓扑名称。

Availability: 1.1

❏

ma_topo 删除 topology.topology 表中的记录, 并删除 geometry_columns 表中的记录。

```
SELECT topology.DropTopology('ma_topo');
```

❏

DropTopoGeometryColumn

9.3.4 RenameTopology

RenameTopology — 重命名拓扑

Synopsis

```
varchar RenameTopology(varchar old_name, varchar new_name);
```

❏

重命名拓扑 schema, 更新其在 topology.topology 表中的元数据记录。

Availability: 3.4.0

¶¶

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(¶¶¶¶¶¶¶¶¶¶)	Identifier of the edge.	Identifier of the node.	An edge has a node in its interior. See ST_Relate .

乱乱	id1	id2	Meaning
invalid edge(乱乱乱乱乱乱乱乱乱乱)	Identifier of the edge.		An edge geometry is invalid. See ST_IsValid .
edge not simple(乱乱乱乱乱乱乱乱乱乱)	Identifier of the edge.		An edge geometry has self-intersections. See ST_IsSimple .
edge crosses edge(乱乱乱乱乱乱乱乱乱乱乱乱乱乱乱乱)	Identifier of first edge.	Identifier of second edge.	Two edges have an interior intersection. See ST_Relate .
edge start node geometry mismatch(乱乱乱乱乱乱乱乱乱乱乱乱乱乱乱乱)	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 ST_StartPoint .
edge end node geometry mismatch(乱乱乱乱乱乱乱乱乱乱乱乱乱乱乱乱)	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 ST_EndPoint .
face without edges(乱乱乱乱乱乱乱乱乱乱)	Identifier of the orphaned face.		No edge reports an existing face on either of its sides (left_face, right_face).
face has no rings(乱乱乱乱乱乱乱乱乱乱)	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 GetRingEdges .		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 GetFaceContainingPoint .
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 GetRingEdges .		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 GetRingEdges .		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 GetRingEdges .	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.

```
SELECT * FROM topology.ValidateTopology('ma_topo');
      error      | id1 | id2
-----+-----+-----
face without edges |    1 |
```

[validatetopology_returntype](#), [Topology_Load_Tiger](#)

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](#)

9.4 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(varchar topology_schema_name);
integer CreateTopology(varchar topology_schema_name, integer srid);
integer CreateTopology(varchar topology_schema_name, integer srid, double precision prec);
integer CreateTopology(varchar topology_schema_name, integer srid, double precision prec, boolean hasz);
```

￼￼

Creates a new topology schema with name `topology_name` and registers it in the `topology.topology` table. Topologies must be uniquely named. The topology tables (`edge_data`, `face`, `node`, and `relation`) are created in the schema. It returns the id of the topology.

The `srid` is the **spatial reference system** SRID for the topology.

The tolerance `prec` is measured in the units of the spatial reference system. The tolerance defaults to 0.

`hasz` defaults to false if not specified.

This is similar to the SQL/MM **ST_InitTopoGeo** but has more functionality.

Availability: 1.1

Enhanced: 2.0 added the signature accepting `hasZ`

￼￼

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 简体中文.

❏

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);


```

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 —  (split)  .

Synopsis

```
integer TopoGeo_AddPoint(varchar atopology, geometry apoint, float8 tolerance);
```



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 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 integer **TopoGeo AddLineString**(varchar atopology, geometry aline, float8 tolerance);

国国

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 国国国国国国国国国国国国.

Enhanced: 3.2.0 added support for returning signed identifier.

国国

[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 integer **TopoGeo_AddPolygon**(varchar atopolgy, geometry apoly, float8 tolerance);

国国

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 国国国国国国国国国国国国.

国国

[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);

注意

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

注意

[TopoGeo_AddPoint](#), [TopoGeo_AddLineString](#), [TopoGeo_AddPolygon](#), [CreateTopology](#)

9.6 拓扑

9.6.1 ST_AddIsoNode

ST_AddIsoNode — 添加 (isolated) 节点 ID 到拓扑。如果 NULL 则不添加，否则添加。

Synopsis

integer **ST_AddIsoNode**(varchar atopology, integer aface, geometry apoint);

注意

atopology 拓扑 aface ID(faceid) 面 ID 节点 ID(nodeid) 节点 ID(nodeid)。

如果 (SRID) 不为 NULL, apoint 不为 NULL 则添加, 否则不添加 (如果为 NULL) 则不添加。如果为 NULL 则不添加。

aface 不为 NULL 则添加 apoint 不为 NULL, 否则不添加。

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X+1.3.1

注意

注意

[AddNode](#), [CreateTopology](#), [DropTopology](#), [ST_Intersects](#)

9.6.2 ST_AddIsoEdge

`ST_AddIsoEdge` — 向拓扑中添加一个节点和另一个节点之间的边。边由 `alinestring` 几何体表示。返回的边 ID 由 `edgeid` 返回。

Synopsis

```
integer ST_AddIsoEdge(varchar atopology, integer anode, integer anothernode, geometry alinestring);
```

返回

向拓扑中添加一个节点和另一个节点之间的边。边由 `alinestring` 几何体表示。返回的边 ID 由 `edgeid` 返回。

`alinestring` 几何体 (SRID) 必须是非 NULL 的，并且必须是有效的几何体，即它必须是有效的几何体。

`alinestring` 必须是一个有效的几何体，即它必须是有效的几何体。

`anode` 和 `anothernode` 必须是一个有效的几何体，即它必须是有效的几何体。

Availability: 1.1

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4

返回

返回

[ST_AddIsoNode](#), [ST_IsSimple](#), [ST_Within](#)

9.6.3 ST_AddEdgeNewFaces

`ST_AddEdgeNewFaces` — 向拓扑中添加新的面。面由 `acurve` 几何体表示。返回的面 ID 由 `faceid` 返回。

Synopsis

```
integer ST_AddEdgeNewFaces(varchar atopology, integer anode, integer anothernode, geometry acurve);
```

返回

向拓扑中添加新的面。面由 `acurve` 几何体表示。返回的面 ID 由 `faceid` 返回。

面必须是有效的几何体，即它必须是有效的几何体。

`acurve` 必须是有效的几何体，即它必须是有效的几何体。返回的面 ID 由 `faceid` 返回。

`acurve` 数据类型 (SRID) 数据类型。

2.0 数据类型。



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` — 数据类型, 数据类型, 数据类型, 数据类型 数据类型。

Synopsis

integer **ST_AddEdgeModFace**(varchar atopology, integer anode, integer anothernode, geometry acurve);

☐☐

数据类型, 数据类型, 数据类型。



Note

数据类型, 数据类型。 数据类型 (数据类型) 数据类型 (universe face) 数据类型。

数据类型 ID 数据类型。

数据类型。

数据类型 NULL 数据类型, 数据类型 (数据类型 node 数据类型), `acurve` 数据类型 `LINESTRING` 数据类型, `anode` 数据类型 `anothernode` 数据类型 数据类型。

`acurve` 数据类型 (SRID) 数据类型。

2.0 数据类型。



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13

☐☐

☐☐

[ST_RemEdgeModFace](#)

[ST_AddEdgeNewFaces](#)

9.6.5 ST_RemEdgeNewFace

ST_RemEdgeNewFace — 移除拓扑中的边，并创建新的面，以修复拓扑。如果边是面的边界，则移除该边并创建新的面。如果边不是面的边界，则移除该边并创建新的面。如果边是面的边界，则移除该边并创建新的面。

Synopsis

integer **ST_RemEdgeNewFace**(varchar atopology, integer anedge);

返回

移除拓扑中的边，并创建新的面，以修复拓扑。如果边是面的边界，则移除该边并创建新的面。如果边不是面的边界，则移除该边并创建新的面。如果边是面的边界，则移除该边并创建新的面。

返回 ID 列表，如果边是面的边界，则返回 NULL。如果边不是面的边界，则返回边 ID。如果边是面的边界，则返回 NULL。如果边不是面的边界，则返回边 ID。如果边是面的边界，则返回 NULL。如果边不是面的边界，则返回边 ID。

如果边是面的边界，则返回 NULL。如果边不是面的边界，则返回边 ID。如果边是面的边界，则返回 NULL。如果边不是面的边界，则返回边 ID。

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).

如果边是 NULL，则返回 NULL。如果边不是 NULL，则返回边 ID。如果边是 NULL，则返回 NULL。如果边不是 NULL，则返回边 ID。如果边是 NULL，则返回 NULL。如果边不是 NULL，则返回边 ID。

2.0 版本引入。

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14

返回

返回

[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

integer **ST_RemEdgeModFace**(varchar atopology, integer anedge);

删除

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.

删除参与定义现有 TopoGeometry 的边。如果删除的边分隔了两个面，则删除其中一个面并修改另一个面以覆盖其空间。优先保留右侧的面，以保持与 **ST_AddEdgeModFace** 一致。返回保留面的 ID。

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).

如果边参与定义现有 TopoGeometry，则拒绝删除该边。如果任何 TopoGeometry 仅由其中一个面定义（而不是另一个），则拒绝愈合两个面。

2.0 版本引入。

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.15

删除

删除

ST_AddEdgeModFace

ST_RemEdgeNewFace

9.6.7 ST_ChangeEdgeGeom

ST_ChangeEdgeGeom — 更改 TopoGeometry 中边的几何形状。

Synopsis

text **ST_ChangeEdgeGeom**(varchar atopology, integer anedge, geometry acurve);

删除

更改 TopoGeometry 中指定边的几何形状。

If any arguments are null, the given edge does not exist in the edge table of the topology schema, the acurve is not a LINESTRING, or the modification would change the underlying topology then an error is thrown.

acurve 指定要更改的边的 SRID。

acurve 必须是 LINESTRING 类型。

如果任何参数为 null，给定的边不存在于拓扑模式中的边表中，acurve 不是 LINESTRING，或者修改将更改底层拓扑，则抛出错误。

1.1.0 版本引入。

更新：2.0.0 版本引入。

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details X.3.6

❏

```
SELECT topology.ST_ChangeEdgeGeom('ma_topo', 1,
    ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6 893844.3,227641.6
    893816.6, 227704.5 893778.5)', 26986) );
----
Edge 1 changed
```

❏

[ST_AddEdgeModFace](#)

[ST_RemEdgeModFace](#)

[ST_ModEdgeSplit](#)

9.6.8 ST_ModEdgeSplit

ST_ModEdgeSplit — Splits an edge in a topology at a point, creating a new edge and a new node. The original edge is replaced by two new edges. The new node is created at the point specified by the geometry argument.

Synopsis

integer **ST_ModEdgeSplit**(varchar atopology, integer anedge, geometry apoint);

❏

ST_ModEdgeSplit splits an edge in a topology at a point, creating a new edge and a new node. The original edge is replaced by two new edges. The new node is created at the point specified by the geometry argument.

Availability: 1.1

Changed in: 2.0 **ST_ModEdgeSplit**, **ST_ModEdgesSplit** **ST_ModEdgesSplit**.

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

❏

```
-- 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;
-----
7
```



[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

```
int ST_ModEdgeHeal(varchar atopology, integer anedge, integer 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

```
int ST_NewEdgeHeal(varchar atopology, integer anedge, integer 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, integer anode, geometry apoint);

返回

返回描述。 apoint 必须是一个点。如果 apoint 已经存在于拓扑中，则抛出异常。

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.

返回描述 (SRID) 必须与现有节点相同。

2.0.0 版本开始。

Enhanced: 3.2.0 ensures the nod cannot be moved in a different face



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.2

返回

```
-- 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
```

返回

ST_AddIsoNode

9.6.12 ST_NewEdgesSplit

ST_NewEdgesSplit — 将拓扑中的边分割成两部分，返回 2 个新的边 ID。如果边 ID 已经存在于拓扑中，则抛出异常。

Synopsis

integer **ST_NewEdgesSplit**(varchar atopology, integer anedge, geometry apoint);

¶¶

ST_Apoint 函数返回一个点几何体，其 ID 为 anedge。该函数接受一个 ID 值。该函数返回的几何体与输入几何体具有相同的 SRID。

ST_Apoint (SRID) 函数返回一个点几何体，其 ID 为 NULL。该函数接受一个 SRID 值。该函数返回的几何体与输入几何体具有相同的 SRID。

Availability: 1.1



This method implements the SQL/MM specification. SQL-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 — 移除拓扑中的孤立节点。该函数接受一个拓扑名称 (varchar) 和一个节点 ID (integer) 作为参数，并返回一个整数。

Synopsis

text **ST_RemoveIsoNode**(varchar atopoology, integer anode);

¶¶

该函数接受一个拓扑名称 (varchar) 和一个节点 ID (integer) 作为参数，并返回一个整数。

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

例

```
-- Remove an isolated node with no face --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7 ) As result;
           result
-----
Isolated node 7 removed
```

例

[ST_AddIsoNode](#)

9.6.14 ST_RemoveIsoEdge

ST_RemoveIsoEdge — Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown.

Synopsis

text **ST_RemoveIsoEdge**(varchar atopology, integer anedge);

例

Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown.

Availability: 1.1

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

例

```
-- Remove an isolated node with no face --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7 ) As result;
           result
-----
Isolated node 7 removed
```

例

[ST_AddIsoNode](#)

9.7 简体中文

9.7.1 GetEdgeByPoint

GetEdgeByPoint — Finds the edge-id of an edge that intersects a given point.

Synopsis

integer **GetEdgeByPoint**(varchar atopology, geometry apoint, float8 tol1);

返回

Retrieves the id of an edge that intersects a Point.

返回类型, 参数, 返回值 (edgeid) 返回类型. tolerance = 0 返回类型
返回类型.

If apoint doesn't intersect an edge, returns 0 (zero).

If use tolerance > 0 and there is more than one edge near the point then an exception is thrown.



Note

返回类型 tolerance = 0 返回 ST_Intersects 返回, 返回类型 ST_DWithin 返回.

GEOS 返回

2.0.0 返回.

返回

返回 **AddEdge** 返回.

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As withlmtol, topology.GetEdgeByPoint('ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('SRID=26986;POINT(227622.6 893843)') As geom;
withlmtol | withnotol
-----+-----
2 | 0
```

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

-- get error --
ERROR: Two or more edges found
```

返回

AddEdge, **GetNodeByPoint**, **GetFaceByPoint**

9.7.2 GetFaceByPoint

GetFaceByPoint — Finds face intersecting a given point.

Synopsis

integer **GetFaceByPoint**(varchar atopology, geometry apoint, float8 tol1);

国国

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 国国国国国国国国国国.

Enhanced: 3.2.0 more efficient implementation and clearer contract, stops working with invalid topologies.

国国

```
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
```

国国

GetFaceContainingPoint, AddFace, GetNodeByPoint, GetEdgeByPoint

9.7.3 GetFaceContainingPoint

GetFaceContainingPoint — Finds the face containing a point.

Synopsis

integer **GetFaceContainingPoint**(text atopology, geometry apoint);

国国

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

integer **GetNodeByPoint**(varchar atopology, geometry apoint, float8 toll1);

☐☐

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 ☐☐☐ ST_Intersects ☐, ☐☐☐☐☐☐☐ ST_DWithin ☐☐☐☐☐☐.

GEOS ☐☐☐☐☐

2.0.0 ☐☐☐☐☐☐☐☐☐☐☐☐☐☐.

☐☐

☐☐☐☐☐☐ [AddEdge](#) ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐.

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
      nearnode
```

```
-----
          2
```

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1000) As too_much_tolerance
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

```
----get error--
ERROR:  Two or more nodes found
```

☐☐

[AddEdge](#), [GetEdgeByPoint](#), [GetFaceByPoint](#)

9.7.5 GetTopologyID

GetTopologyID — 返回 topology.topology 的 ID。

Synopsis

integer **GetTopologyID**(varchar toponame);

返回

返回 topology.topology 的 ID。

Availability: 1.1

返回

```
SELECT topology.GetTopologyID('ma_topo') As topo_id;
topo_id
-----
1
```

返回

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologySRID](#)

9.7.6 GetTopologySRID

GetTopologySRID — 返回 topology.topology 的 SRID。

Synopsis

integer **GetTopologyID**(varchar toponame);

返回

返回 topology.topology 的 SRID。
2.0.0 版本中可用。

返回

```
SELECT topology.GetTopologySRID('ma_topo') As SRID;
SRID
-----
4326
```

函数

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologyID](#)

9.7.7 GetTopologyName

`GetTopologyName` — 给定 ID 的拓扑 (名称) 名称。

Synopsis

`varchar` **GetTopologyName**(integer topology_id);

函数

给定 ID 的 topology.topology 名称 (名称) 名称。

Availability: 1.1

函数

```
SELECT topology.GetTopologyName(1) As topo_name;
topo_name
-----
ma_topo
```

函数

[CreateTopology](#), [DropTopology](#), [GetTopologyID](#), [GetTopologySRID](#)

9.7.8 ST_GetFaceEdges

`ST_GetFaceEdges` — 给定 aface 的拓扑面边缘。

Synopsis

`getfaceedges_returntype` **ST_GetFaceEdges**(varchar atopology, integer aface);

函数

`aface` 给定拓扑面的 ID。返回的序列 (sequence) 给定 ID(edgeid) 的边 ID。返回 1 个边 ID。

返回的序列 (sequence) 给定 ID(edgeid) 的边 ID。返回 1 个边 ID。

2.0 版本。



This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.5

SQL

```
-- 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;
```

SQL

GetRingEdges, AddFace, ST_GetFaceGeometry

9.7.9 ST_GetFaceGeometry

ST_GetFaceGeometry — Returns the geometry of the face with ID.

Synopsis

geometry **ST_GetFaceGeometry**(varchar atopology, integer aface);

SQL

Returns the geometry of the face with ID. Availability: 1.1

 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.16

SQL

```
-- 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,
```



9.7.10 GetRingEdges

Synopsis



```
max_edges = NULL;
XXXXXXXXXXXXXXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXX.
```

[illegible]

9.7.11 GetNodeEdges

Synopsis

```
getfaceedges returntype GetNodeEdges(varchar atopology, integer anode);
```


9.8.2 AddNode

AddNode — 向拓扑中添加节点。向拓扑添加节点时，指定 ID(nodeid) 和点几何。指定 ID 时，ID 必须唯一。

Synopsis

integer **AddNode**(varchar toponame, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);

参数

toponame 是拓扑名称。AddEdge 向拓扑添加边时，指定 ID(edgeid) 和线几何。指定 ID 时，ID 必须唯一。

allowEdgeSplitting 指定是否允许边分裂。如果为 true，则允许边分裂。如果为 false，则不允许边分裂。

computeContainingFace 指定是否计算包含面。如果为 true，则计算包含面。如果为 false，则不计算包含面。



Note

apoint 是点几何。指定 ID(nodeid) 时，ID 必须唯一。

2.0.0 版本引入。

示例

```
SELECT topology.AddNode('ma_topo', ST_GeomFromText('POINT(227641.6 893816.5)', 26986) ) As  
       nodeid;  
-- result --  
nodeid  
-----  
4
```

示例

AddEdge, CreateTopology

9.8.3 AddEdge

AddEdge — 向拓扑中添加边。向拓扑添加边时，指定 ID(edgeid) 和线几何。指定 ID 时，ID 必须唯一。

Synopsis

integer **AddEdge**(varchar toponame, geometry aline);

名称 (toponame) 为字符串, 表示该边缘的
 名称 (名称) 为字符串 ID(edgeid) 为字符串. 该边缘的
 (universe) 为字符串.

Note!

Note

$\frac{1}{2} \text{aline}$

Note!

Note

aline srid srid .

GEOS ☒ ☒ ☒ ☒ ☒



Warning

AddEdge is deprecated as of 3.5.0. Use **TopoGeo AddLineString** instead.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.



```
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
```



TopoGeo AddLineString, CreateTopology, Section 4.5


```

) 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 — - (Douglas-Peucker) TopoGeometry

Synopsis

```
geometry ST_Simplify(TopoGeometry tg, float8 tolerance);
```


TopoGeometry (Douglas-Peucker) TopoGeometry " "



Note

[illegible]

GEOS ☒☒☒☒☒

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.

国国

国国 [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

```
int RemoveUnusedPrimitives(text topology_name, geometry bbox);
```

国国

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

国国

[ST_ModEdgeHeal](#), [ST_RemEdgeModFace](#)

9.9 TopoGeometry 国国国

9.9.1 CreateTopoGeom

CreateTopoGeom — 国国国国国国国国国国国国国国国国国国国国国国. `tg_type` 国 1: [国国] 国国国, 2: [国国] 国国, 3: [国国] 国国国, 4: 国国国国国国.

Synopsis

```
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id, topoelementarray tg_objs);
```

```
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id);
```

国国

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

 :

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);
```

TopoGeometry

1. `blockgroups` 是 `TopoGeometry` 的。

```
-- 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;
```



AddTopoGeometryColumn, toTopoGeom ST_CreateTopoGeo, ST_GetFaceGeometry, TopoElementArray, TopoElementArray Agg

9.9.2 toTopoGeom

`toTopoGeom` — Converts a simple Geometry into a topo geometry.

Synopsis

```
topogeometry toTopoGeom(geometry geom, varchar toponame, integer layer_id, float8 tolerance);
topogeometry toTopoGeom(geometry geom, topogeometry topogeom, float8 tolerance);
```



TopoGeometry.

The `relation` is `TopoGeometry`.

TopoGeometry (topogeom) [TopoGeometry \(topogeom\)](#)

tolerance ██.

1 (toponame) (layer id) TopoGeometry .

[illegible]

2.0 ☒☒☒☒☒☒☒☒☒☒☒.

TopoGeometry 2.1.0



XXXXXXXXXXXXXXXXXXXXXXXX (workflow) XXXX.

```
-- do this if you don't have a topology setup already
-- creates topology not allowing any tolerance
SELECT topology.CreateTopology('topo_boston_test', 2249);
-- create a new table
CREATE TABLE nei_topo(gid serial primary key, nei varchar(30));
--add a topogeometry column to it
SELECT topology.AddTopoGeometryColumn('topo_boston_test', 'public', 'nei_topo', 'topo', ' ←
MULTIPOLYGON') As new layer id;
```

```

new_layer_id
-----
1

--use new layer id in populating the new topogeometry column
-- we add the topogeoms to the new layer with 0 tolerance
INSERT INTO nei_topo(nei, topo)
SELECT nei, topology.toTopoGeom(geom, 'topo_boston_test', 1)
FROM neighborhoods
WHERE gid BETWEEN 1 and 15;

--use to verify what has happened --
SELECT * FROM
    topology.TopologySummary('topo_boston_test');

-- summary--
Topology topo_boston_test (5), SRID 2249, precision 0
61 nodes, 87 edges, 35 faces, 15 topogeoms in 1 layers
Layer 1, type Polygonal (3), 15 topogeoms
Deploy: public.nei_topo.topo

-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = 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

setof topoelement **GetTopoGeomElements**(varchar toponame, integer layer_id, integer 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 Ⓜ layer_id Ⓜ 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
```


[illegible]

```
SELECT topology.AsGML(topo,'') As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';

-- rdgml--
<TopoCurve>
  <directedEdge>
    <Edge id="E1">
      <directedNode orientation="-">
        <Node id="N1"/>
      </directedNode>
      <directedNode>
    ></directedNode>
    <curveProperty>
      <Curve srsName="urn:ogc:def:crs:EPSG::3438">
        <segments>
```

```

<LineStringSegment>
  <posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
    384833 236884 384844 236882 384866 236881 384879 236883 384954 ↵
      236898 385087 236932 385117 236938
    385167 236938 385203 236941 385224 236946 385233 236950 385241 ↵
      236956 385254 236971
    385260 236979 385268 236999 385273 237018 385273 237037 385271 ↵
      237047 385267 237057 385225 237125
    385210 237144 385192 237161 385167 237192 385162 237202 385159 ↵
      237214 385159 237227 385162 237241
    385166 237256 385196 237324 385209 237345 385234 237375 385237 ↵
      237383 385238 237399 385236 237407
    385227 237419 385213 237430 385193 237439 385174 237451 385170 ↵
      237455 385169 237460 385171 237475
    385181 237503 385190 237521 385200 237533 385206 237538 385213 ↵
      237541 385221 237542 385235 237540 385242 237541
    385249 237544 385260 237555 385270 237570 385289 237584 385292 ↵
      237589 385291 237596 385284 237630</posList>
  </LineStringSegment>
</segments>
</Curve>
</curveProperty>
</Edge>
</directedEdge>
</TopoCurve>

```



CreateTopoGeom, ST_CreateTopoGeo

9.12.2 AsTopoJSON

AsTopoJSON — TopoGeometry  TopoJSON                   .

Synopsis

```
text AsTopoJSON(topogeometry tg, regclass edgeMapTable);
```

[illegible]

XXXXXXXXXX, XXXX" XX (serial)" XX"arc id" XXXXXX"edge id" XXXXXXXXXXXXXXXX.
 XX"edge id" XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.



Note

TopoJSON 0-1 "edgeMapTable" 1-1.

TopoJSON 格式, 该格式是 GeoJSON (snippet) 格式, 该格式是 GeoJSON 格式。
 TopoJSON 格式是 GeoJSON 格式。

2.1.0 格式。

格式: 2.2.1 格式 (puntal) 格式。

格式

ST_AsGeoJSON

格式

```
CREATE TEMP TABLE edgemap(arc_id serial, edge_id int unique);

-- header
SELECT '{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects" ←
      ": {'

-- objects
UNION ALL SELECT '' || feature_name || ': ' || AsTopoJSON(feature, 'edgemap')
FROM features.big_parcel WHERE feature_name = 'P3P4';

-- arcs
WITH edges AS (
  SELECT m.arc_id, e.geom FROM edgemap m, city_data.edge e
  WHERE e.edge_id = m.edge_id
), points AS (
  SELECT arc_id, (st_dumppoints(geom)).* FROM edges
), compare AS (
  SELECT p2.arc_id,
    CASE WHEN p1.path IS NULL THEN p2.geom
    ELSE ST_Translate(p2.geom, -ST_X(p1.geom), -ST_Y(p1.geom))
  END AS geom
  FROM points p2 LEFT OUTER JOIN points p1
  ON ( p1.arc_id = p2.arc_id AND p2.path[1] = p1.path[1]+1 )
  ORDER BY arc_id, p2.path
), arcdump AS (
  SELECT arc_id, (regexp_matches( ST_AsGeoJSON(geom), '\[.*\]'))[1] as t
  FROM compare
), arcs AS (
  SELECT arc_id, '[' || array_to_string(array_agg(t), ',') || ']' as a FROM arcdump
  GROUP BY arc_id
  ORDER BY arc_id
)
SELECT '}', "arcs": [' UNION ALL
SELECT array_to_string(array_agg(a), E'\n') from arcs

-- footer
UNION ALL SELECT '}]':::text as t;

-- Result:
{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects": {
"P3P4": { "type": "MultiPolygon", "arcs": [[[ -1],[6,5,-5,-4,-3,1]]]
}, "arcs": [
[[25,30],[6,0],[0,10],[-14,0],[0,-10],[8,0]],
[[35,6],[0,8]],
[[35,6],[12,0]],
[[47,6],[0,8]],
[[47,14],[0,8]],
```

```
[[35,22],[12,0]],
[[35,14],[0,8]]
]}
```

9.13 空间拓扑

9.13.1 Equals

Equals — 检查 TopoGeometry 是否相等。

Synopsis

boolean **Equals**(topogeometry tg1, topogeometry tg2);

返回

检查 TopoGeometry 是否相等 (点, 线, 面)。



Note
检查 TopoGeometry 是否相等。检查 TopoGeometry 是否相等。

1.1.0 版本。

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

返回

返回

[GetTopoGeomElements](#), [ST_Equals](#)

9.13.2 Intersects

Intersects — 检查 TopoGeometry 是否相交。

Synopsis

boolean **Intersects**(topogeometry tg1, topogeometry tg2);

11

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

栅格数据, 栅格索引

10.1 栅格数据

栅格数据, 栅格索引 raster2pgsql 是 PostGIS 的一个扩展。

10.1.1 raster2pgsql 栅格数据

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

栅格数据 (factor) 栅格数据, 栅格索引 <http://trac.osgeo.org/postgis/ticket/1764> 栅格数据

10.1.1.1 Example Usage

栅格数据 100x100 栅格数据:

```
# -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.

[illegible]

```
raster2pgsql -s 4326 -I -C -M *.tif -F -t 100x100 public.demelevation | psql -d gisdb
```

```

aerial 2 4
(DB )
-e (
). 128x128
-F "filename"
.

```

```
raster2pgsql -I -C -e -Y -F -s 26986 -t 128x128 -l 2,4 bostonaerials2008/*.jpg aerials. ↵
boston | psql -U postgres -d gisdb -h localhost -p 5432
```

```
--get a list of raster types supported:
```

```
raster2pgsql -G
```

-G □□□□□□□□□□□□□□□□:

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

-? □□□□□□□□□□. □□□□□□□□□□□□□□□□□□□□□□.

-G ☒☒☒☒☒☒☒☒☒☒☒☒☒.

[illegible]

-C □□□□□□□□□□ (X) □□□□□□□□□□. □□□□□□□□□□.

-a ☐☐☐☐☐☐☐☐☐ (☐) ☐☐☐☐☐☐.

-d , () .

-p ☒☒☒☒☒, ☒☒☒☒☒☒☒☒.

[illegible]

```
-C raster_columns [columns] SRID, [columns]
```

-x `XXXXXXXX` (extent) `XXXXXXXXXXXXXXXXXXXX`. **-C** `XXXXXXXXXXXXXXXXXXXXXXXXXXXX`.

-r **regular blocking** 使用常规阻塞 (regular blocking) 模式。-C 选项
指定并行度。

选项: 指定要写入的表名。

-s <SRID> 指定 SRID 值。如果为 0，则使用 SRID 值。
指定要写入的表名。

-b BAND 指定要写入的波段 (1-255)。如果为 0，则使用所有波段。
指定要写入的表名。

-t TILE_SIZE 指定要写入的瓦片大小。TILE_SIZE 指定为 x 或 y
指定要写入的表名。

-P 指定要写入的瓦片大小 (padding) 指定要写入的表名。

-R, --register 指定要写入的表名 (DB 表) 指定要写入的表名。
指定要写入的表名。

-l OVERVIEW_FACTOR 指定要写入的表名。指定要写入的表名。
指定要写入的表名。

-N NODATA "NODATA" 指定要写入的表名 NODATA 指定要写入的表名。

选项: 指定要写入的表名。

-f COLUMN 指定要写入的表名。指定要写入的表名。

-F 指定要写入的表名。

-n COLUMN 指定要写入的表名。-F 指定要写入的表名。

-q PostgreSQL 指定要写入的表名。

-I 指定要写入的表名。

-M 指定要写入的表名 (vacuum analyze) 指定要写入的表名。

-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 指定要写入的表名。-X 指定要写入的表名 (指定要写入的表名)

-X tablespace 指定要写入的表名。-I 指定要写入的表名。

-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 指定要写入的表名, 指定要写入的表名 (transaction) 指定要写入的表名。

-E ENDIAN 指定要写入的表名 (endianness) 指定要写入的表名。XDR 0, 指定
指定要写入的表名。指定要写入的表名。

-V version 指定要写入的表名。指定要写入的表名。指定要写入的表名。

10.1.2 PostGIS 简体中文版

指定要写入的表名。指定要写入的表名。

1. 指定要写入的表名:

```
CREATE TABLE myrasters(rid serial primary key, rast raster);
```

2. 创建空栅格。使用 `ST_MakeEmptyRaster` 函数，`ST_AddBand` 函数添加波段。
使用 `ST_AsRaster` 函数，`ST_Union`，`ST_MapAlgebra` (algebra) 函数。
使用 `ST_Transform` 函数。
3. 创建索引，使用 `ST_ConvexHull` 函数：

```
CREATE INDEX myrasters_rast_st_convexhull_idx ON myrasters USING gist( ST_ConvexHull( ←  
rast) );
```

(convex hull) 使用 `ST_ConvexHull` 函数。



Note

PostGIS 2.0 使用 `envelop` 函数。
使用 `ST_ConvexHull` 函数。

4. `AddRasterConstraints` 函数。

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=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \  
raster2pgsql \  
-s 990000 \  
-t 256x256 \  
-I \  
-R \  
/vsis3/your.bucket.com/your_file.tif \  
your_table \  
| psql your_db
```

Once the table is loaded, you need to give the database permission to read from remote rasters, by setting two permissions, `postgis.enable_outdb_rasters` and `postgis.gdal_enabled_drivers`.

```
SET postgis.enable_outdb_rasters = true;  
SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

To make the changes sticky, set them directly on your database. You will need to re-connect to experience the new settings.

```
ALTER DATABASE your_db SET postgis.enable_outdb_rasters = true;
ALTER DATABASE your_db SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

For non-public rasters, you may have to provide access keys to read from the cloud rasters. The same keys you used to write the `raster2pgsql` call can be set for use inside the database, with the `postgis.gdal_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 云栅格

PostGIS 支持云栅格。云栅格是指存储在云存储（如 Amazon S3）中的栅格数据。PostGIS 通过 GDAL 的 VSI 接口与云存储交互。云栅格可以是公共的，也可以是私有的。对于私有栅格，您需要提供访问密钥来读取数据。

1. `raster_columns` 表用于存储栅格元数据。您可以通过 `raster_columns` 表来管理栅格数据。
2. `raster_overviews` 表用于存储栅格概览数据。您可以通过 `raster_overviews` 表来管理栅格概览数据。

10.2.1 栅格元数据

`raster_columns` 表用于存储栅格元数据。您可以通过 `raster_columns` 表来管理栅格数据。该表包含以下列：
`raster_columns` 表包含以下列：
`raster_columns` 表包含以下列：

您可以通过 `raster_columns` 表来管理栅格数据。您可以通过 `raster_columns` 表来管理栅格数据。您可以通过 `raster_columns` 表来管理栅格数据。

- `r_table_catalog` 栅格表所属的数据库名称。
- `r_table_schema` 栅格表所属的模式名称。
- `r_table_name` 栅格表的名称。
- `r_raster_column` 栅格表的列名。PostGIS 会自动为栅格表添加 `AddRasterConstraints` 约束。
- `srid` 栅格数据的 SRID。Section 4.5 介绍了 SRID 的使用。
- `scale_x` 栅格数据的 X 轴比例尺。您可以通过 `scale_x` 来设置比例尺。

- `scale_y` 縦方向のスケール (倍率) を指定。デフォルトは `scale_y` を 1.0 に設定。 `ST_ScaleY` を参照。
- `blocksize_x` 横方向のブロックサイズ (ピクセル) を指定。デフォルトは `ST_Width` を参照。
- `blocksize_y` 縦方向のブロックサイズ (ピクセル) を指定。デフォルトは `ST_Height` を参照。
- `same_alignment` 出力の出力位置を指定。デフォルトは `ST_SameAlignment` を参照。
- `regular_blocking` 出力の出力位置を指定。デフォルトは `ST_SameAlignment` を参照。
- `num_bands` 出力の出力位置を指定。デフォルトは `ST_NumBands` を参照。
- `pixel_types` 出力の出力位置を指定。デフォルトは `ST_BandPixelType` を参照。
- `nodata_values` `nodata_value` (double precision) を指定 (数) を指定。デフォルトは `ST_BandNoDataValue` を参照。
- `out_db` 出力の出力位置を指定。デフォルトは `ST_SameAlignment` を参照。
- `extent` 出力の出力位置を指定 (extent) を指定。デフォルトは `DropRasterConstraints` を参照。
- `spatial index` 出力の出力位置を指定。

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 .

- `o_table_catalog` 表所在数据库名称。
- `o_table_schema` 表所在模式名称。
- `o_table_name` 表名称。
- `o_raster_column` 栅格列名称。
- `r_table_catalog` 表所在数据库名称。
- `r_table_schema` 表所在模式名称。
- `r_table_name` 表名称。
- `r_raster_column` 栅格列名称。
- `overview_factor` 概览因子。栅格被划分为 $2^{\text{overview\_factor}}$ 个部分。栅格2pgsql 将栅格划分为 1 个部分。栅格 2 被划分为 4 个部分。栅格被划分为 125x125 个部分。5000x5000 栅格被划分为 1600 个部分，栅格 `o_2` ($l=2$) 被划分为 $(1600/2^2) = 400$ 个部分，`o_3` ($l=3$) 被划分为 $(1600/2^3) = 200$ 个部分。栅格被划分为 125x125 个部分，栅格 (scrap) 被划分为 125x125 个部分。栅格2pgsql 将栅格划分为 125x125 个部分，栅格被划分为 125x125 个部分 ($2^{\text{overview\_factor}}$) 个部分。

10.3 PostGIS 栅格渲染

The fact that PostGIS raster provides you with SQL functions to render rasters in known image formats gives you a lot of options for rendering them. For example you can use OpenOffice / LibreOffice for rendering as demonstrated in [Rendering PostGIS Raster graphics with LibreOffice Base Reports](#). In addition you can use a wide variety of languages as demonstrated in this section.

10.3.1 使用 ST_AsPNG 在 PHP 中渲染栅格

在 PHP 中，PHP PostgreSQL 扩展提供了 `ST_AsGDALRaster` 函数，可以渲染 1, 2, 3 波段栅格。PHP 扩展 (request stream) 提供了 `ST_AsPNG` 函数。在 PHP 中，使用 `"img src"` HTML 属性来渲染栅格。

在 PHP 中，使用 `ST_AsPNG` 函数 (combine) 渲染 WGS84 栅格。使用 `ST_Union` 函数 (union) 渲染栅格，使用 `ST_Transform` 函数渲染栅格，使用 `ST_AsPNG` 函数渲染 PNG 格式栅格。

http://mywebserver/test_raster.php?srid=2249

在 PHP 中，使用 `ST_AsPNG` 函数渲染栅格。

```
<?php
/** contents of test_raster.php */
$conn_str = 'dbname=mydb host=localhost port=5432 user=myuser password=mypwd';
$dbconn = pg_connect($conn_str);
header('Content-Type: image/png');
/**If a particular projection was requested use it otherwise use mass state plane meters ←
**/
if (!empty( $_REQUEST['srid'] ) && is_numeric( $_REQUEST['srid'] )) {
```

```

        $input_srid = intval($_REQUEST['srid']);
    }
    else { $input_srid = 26986; }
    /** The set bytea_output may be needed for PostgreSQL 9.0+, but not for 8.4 */
    $sql = "set bytea_output='escape';
    SELECT ST_AsPNG(ST_Transform(
        ST_AddBand(ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast,3)]),
        ,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) 用于在 HTML 中显示。PHP 使用 `img src` HTML 属性来显示栅格。

在 npgsql PostgreSQL .NET 中使用 <http://npgsql.projects.postgresql.org/> 提供的 ASP.NET bin 文件。

在 npgsql PostgreSQL .NET 中使用 **ST_Union** (union) 函数，**ST_Transform** 函数，**ST_AsPNG** 函数来生成 PNG 栅格。

在 C# 中使用 Section 10.3.1 中的代码。

```
http://mywebserver/test_raster.php?srid=2249
```

在 web.config 文件中添加以下配置。

```

-- 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 使用 Npgsql 在 Java 中访问 PostGIS

本节将介绍如何在 Java 中使用 Npgsql 访问 PostGIS。

<http://jdbc.postgresql.org/download.html> 提供了 PostgreSQL JDBC 驱动程序的下载链接。

安装驱动程序后，可以在 Java 代码中使用：


```
set env CLASSPATH ../../postgresql-9.0-801.jdbc4.jar
javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class
```

编译并打包后的文件如下：

```
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 与 SQL 交互

PLPython 是 PostgreSQL 的一个扩展。PLPython 是 PostgreSQL 的一个扩展。PLPython 是 PostgreSQL 的一个扩展。PLPython 是 PostgreSQL 的一个扩展。

```

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 与 PostgreSQL 交互

PostgreSQL 是一个开源的数据库系统。PostgreSQL 是一个开源的数据库系统。PostgreSQL 是一个开源的数据库系统。PostgreSQL 是一个开源的数据库系统。

PostgreSQL 是一个开源的数据库系统。PostgreSQL 是一个开源的数据库系统。PostgreSQL 是一个开源的数据库系统。PostgreSQL 是一个开源的数据库系统。

```

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);
```



```
CREATE TABLE dummy_rast(rid integer, rast raster);
INSERT INTO dummy_rast(rid, rast)
VALUES (1,
('01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0000' -- nBands (uint16 0)
||
'000000000000000040' -- scaleX (float64 2)
||
'0000000000000000840' -- scaleY (float64 3)
||
'000000000000E03F' -- ipX (float64 0.5)
||
'000000000000E03F' -- ipY (float64 0.5)
||
'0000000000000000' -- skewX (float64 0)
||
'0000000000000000' -- skewY (float64 0)
||
'00000000' -- SRID (int32 0)
||
'0A00' -- width (uint16 10)
||
'1400' -- height (uint16 20)
)::rast
),
-- Raster: 5 x 5 pixels, 3 bands, PT_8BUI pixel type, NODATA = 0
(2, ('01000003009A9999999999A93F9A999999999A9BF000000E02B274A' ||
'410000000077195641000000000000000000000000000000000000000000000000 ←
FFFFFFFFFF050005000400FDFFDFEFDFEFDFEFDF9FAFEF' ||
' ←
EFCF9FBFDFEFDFCFEFEFEFE04004E627AADD16076B4F9FE6370A9F5FE59637AB0E54F58617087040046566487A1506
')::rast);
```

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 值, 返回 geometry。

pixeltype text **ST_BandPixelType** 返回 geometry。

initialvalue double precision 返回 geometry。

nodataval double precision 返回 NODATA 值。 NULL 值, 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。

栅格

ST_MapAlgebra (callback function version)

11.1.4 raster

raster — 栅格数据类型。

栅格

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 与 PostGIS 集成。栅格数据可以存储在 PostGIS 数据库中，
使用 `ST_ConvexHull` 函数。栅格数据可以存储在 PostGIS 数据库中，
使用 `ST_ConvexHull` 函数。

栅格数据

栅格数据可以存储在 PostGIS 数据库中，使用 `ST_ConvexHull` 函数。

栅格数据	栅格
栅格	栅格

栅格

Chapter 11

11.1.5 reclassarg

reclassarg — 栅格数据类型 ST_Reclass 函数。

栅格

栅格数据类型 ST_Reclass 函数。

nband integer 栅格数据类型。

reclassexpr text 栅格数据类型 range:map range 栅格数据类型。': ' 栅格数据类型
栅格数据类型。'(' 栅格数据类型, ') ' 栅格数据类型, ']' 栅格数据类型
栅格数据类型, '[' 栅格数据类型。

1. [a-b] = a <= x <= b
2. (a-b) = a < x <= b
3. [a-b) = a <= x < b

4. $(a-b) = a < x < b$

'(' 返回 (a-b) 与 a-b 的交集。

pixeltype text ST_BandPixelType 返回像素类型。

nodataval double precision NODATA 值。返回 NODATA 值，或
返回 NODATA 值。

例: 2 与 255 与 NODATA 与 8BUI 的交集。

```
SELECT ROW(2, '0-100:1-10, 101-500:11-150,501 - 10000: 151-254', '8BUI', 255)::reclassarg;
```

例: 1 与 NODATA 与 1BB 的交集。

```
SELECT ROW(1, '0-100]:0, (100-255:1', '1BB', NULL)::reclassarg;
```

例

ST_Reclass

11.1.6 summarystats

summarystats — ST_SummaryStats 与 ST_SummaryStatsAgg 返回统计摘要。

例

ST_SummaryStats 与 **ST_SummaryStatsAgg** 返回统计摘要。

count integer 返回计数。

sum double precision 返回总和。

mean double precision 返回平均值。

stddev double precision 返回标准差。

min double precision 返回最小值。

max double precision 返回最大值。

例

ST_SummaryStats, ST_SummaryStatsAgg

11.1.7 unionarg

unionarg — 返回 UNION 的 ST_Union 结果。

例

```
CREATE TABLE test UNION SELECT ST_Union (geom) FROM test;
```

nband integer 指定栅格列的波段数，默认为 1。

uniontype text UNION 操作的结果类型。 **ST_Union** 操作的结果类型为 **geometry**。

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 查看栅格表中的列名，并返回栅格表的名称。
rastschema 指定栅格表的模式名。 srid 指定 SPATIAL_REF_SYS 表中的 SRID。
extent 指定栅格表的 extent。
```

```
raster2pgsql 将栅格数据加载到数据库中。
```

有关栅格函数的详细信息，请参见 Section 10.2.1 栅格函数。

- **blocksize** X Y 指定栅格表的块大小。
- **blocksize_x** X 指定 (X 轴方向) 的块大小。
- **blocksize_y** Y 指定 (Y 轴方向) 的块大小。
- **extent** 指定栅格表的 extent。指定栅格表的 extent。
- **num_bands** 指定栅格表的波段数。

- `pixel_types` 指定像素数据类型。指定 N 指定像素数据类型。
- `regular_blocking` 指定 (指定) 指定。
- `same_alignment` ensures they all have same alignment meaning any two tiles you compare will return true for. Refer to [ST_SameAlignment](#).
- `srid` 指定 SRID 指定。
- `--` 指定。



Note

指定。指定, 指定。



Note

指定, 指定。
`DropRasterConstraints` 指定。

2.0.0 文档。

例: 文档

```
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}
```

例: 文档

```
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 — [PostGIS 2.1.11](#). [PostGIS 2.1.11](#) [PostGIS 2.1.11](#).

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);

```

[illegible][illegible]

```
DROP TABLE mytable
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXX, XX SQL XXXXXXXX.

```
ALTER TABLE mytable DROP COLUMN rast
```

```

raster_columns raster_columns.
raster_columns raster_columns.

```

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

❏

PostGIS 使用 GDAL 库。GDAL 库的版本信息存储在 `postgis_gdal_version` 表中。

❏

```
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` — 返回 PostGIS 构建 Raster 库的日期。

Synopsis

text **PostGIS_Raster_Lib_Build_Date()**;

❏

返回日期。

❏

```
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` — 返回 PostGIS 构建 Raster 库的版本。

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

￼￼: ￼￼￼￼￼￼￼￼￼￼￼

```
-- 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). 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			
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 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);

￼

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

￼

```
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;
```

￼

[ST_InterpolateRaster](#)

11.2.10 ST_InterpolateRaster

ST_InterpolateRaster — Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation.

Synopsis

raster **ST_InterpolateRaster**(geometry input_points, text algorithm_options, raster template, integer template_band_num=1);

Description

Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation. There are five interpolation algorithms available: inverse distance, inverse distance nearest-neighbor, moving average, nearest neighbor, and linear interpolation. See the [gdal_grid documentation](#) for more details on the algorithms and their parameters. For more information on how interpolations are calculated, see the [GDAL grid tutorial](#).

Input parameters are:

input_points The points to drive the interpolation. Any geometry with Z-values is acceptable, all points in the input will be used.

algorithm_options A string defining the algorithm and algorithm options, in the format used by [gdal_grid](#). For example, for an inverse-distance interpolation with a smoothing of 2, you would use "invdist:smoothing=2.0"

template A raster template to drive the geometry of the output raster. The width, height, pixel size, spatial extent and pixel type will be read from this template.

template_band_num By default the first band in the template raster is used to drive the output raster, but that can be adjusted with this parameter.

Availability: 3.2.0

Example

```
SELECT ST_InterpolateRaster(
  'MULTIPOINT(10.5 9.5 1000, 11.5 8.5 1000, 10.5 8.5 500, 11.5 9.5 500)::geometry,
  'invdist:smoothing:2.0',
  ST_AddBand(ST_MakeEmptyRaster(200, 400, 10, 10, 0.01, -0.005, 0, 0), '16BSI')
)
```

See Also

[ST_Contour](#)

11.2.11 UpdateRasterSRID

UpdateRasterSRID — [Update the SRID of a raster](#).

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

将指定栅格列的 SRID 更新为指定的 SRID。如果指定了 schema_name，则更新 schema_name 中的 table_name 表中的 column_name 列。如果未指定 schema_name，则更新当前数据库中的 table_name 表中的 column_name 列。



Note

更新后的 SRID 必须与原始 SRID 兼容。否则，将引发错误。

2.1.0 版本引入。

更新

UpdateGeometrySRID

11.2.12 ST_CreateOverview

ST_CreateOverview — 创建栅格概览。

Synopsis

regclass **ST_CreateOverview**(regclass tab, name col, int factor, text algo='NearestNeighbor');

更新

创建指定栅格列的概览。概览的分辨率是原始栅格的 1/factor 倍。如果指定了 schema_name，则创建 schema_name 中的 table_name 表中的 column_name 列。如果未指定 schema_name，则创建当前数据库中的 table_name 表中的 column_name 列。

raster_overviews 表将包含所有创建的概览。默认情况下，概览将使用 'NearestNeighbor' 算法。

支持的算法包括 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', 以及 'Lanczos'。有关更多选项，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);
```

例

[ST_Retile](#), [AddOverviewConstraints](#), [AddRasterConstraints](#), [Section 10.2.2](#)

11.3 関数 (constructor)

11.3.1 ST_AddBand

ST_AddBand — 指定した位置 (index) に指定したタイプ (type) の新しいバンドを追加する。初期値 (initialvalue) とノodata値 (nodataval) を指定する。

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);

例

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 例, **addbandarg** 例

5 例, **torast** NULL 例 **fromband** 例 (累計) 例。

outdbfile 6 7 例, **outdbfile** 例。 PostgreSQL 例。

例: 2.1.0 例 **addbandarg** 例。

例: 2.1.0 例 DB 例。

例: 例

```
-- 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);

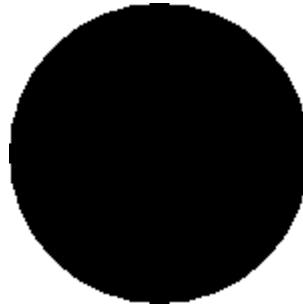


2.0.0 000000000000. GDAL 1.6.0 000000000000.

**Note**

ST_AsPNG, TIN, ST_GeomFromTIN, ST_GeomFromWKB, GDAL 驱动程序。
ST_AsPNG 和 ST_AsJPEG 使用 GDAL 驱动程序。

示例: PNG 输出



示例

```
-- 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 示例

```
-- 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]));
```

示例

[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.

Synopsis

```
raster ST_Band(raster rast, integer[] nbands = ARRAY[1]);
raster ST_Band(raster rast, integer nband);
raster ST_Band(raster rast, text nbands, character delimiter=,);
```



Returns one or more bands of an existing raster as a new raster. Useful for building new rasters from existing rasters or export of only selected bands of a raster or rearranging the order of bands in a raster. If no band is specified or any of specified bands does not exist in the raster, then all bands are returned. Used as a helper function in various functions such as for deleting a band.

Warning



```
nbands = 3, dtype='uint8', nodata=-1, name='1,2,3')
ST_Band(rast, '1@2@3', '@')
ST_Band(rast, '{1,2,3}':int());
PostGIS text
```

2.0.0 □□□□□□□□□□.

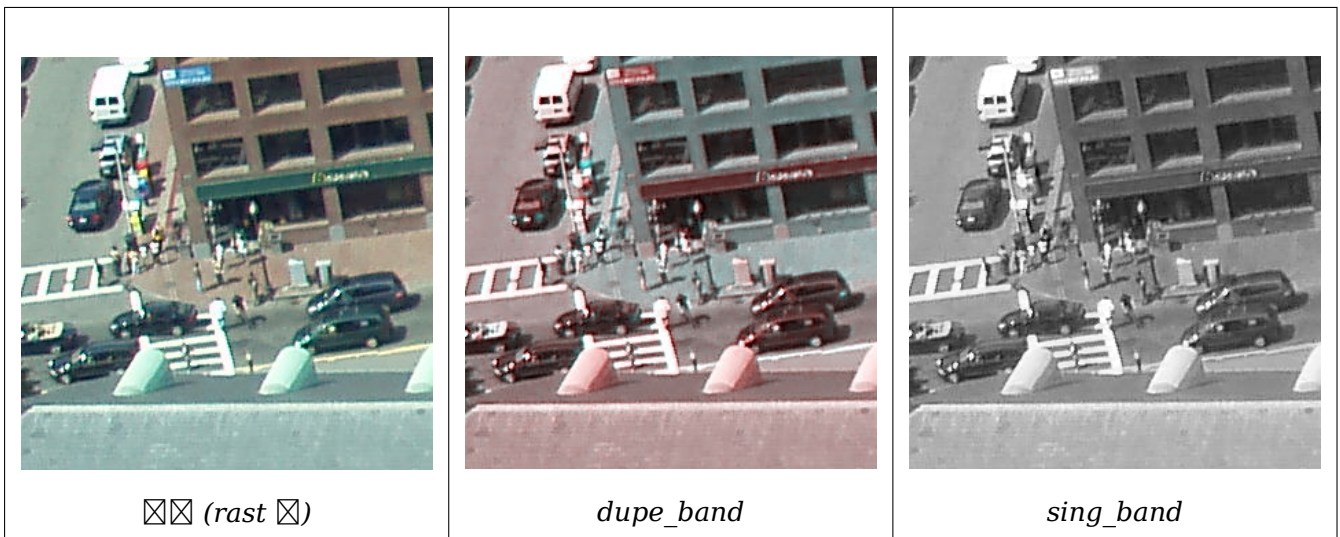

```
-- Make 2 new rasters: 1 containing band 1 of dummy, second containing band 2 of dummy and
    then reclassified as a 2BUI
SELECT ST_NumBands(rast1) As numb1, ST_BandPixelType(rast1) As pix1,
    ST_NumBands(rast2) As numb2, ST_BandPixelType(rast2) As pix2
FROM (
    SELECT ST_Band(rast) As rast1, ST_Reclass(ST_Band(rast,3), '100-200):1, [200-254:2', '2
        BUI') As rast2
        FROM dummy_rast
        WHERE rid = 2) As foo;
```

numb1	pix1	numb2	pix2
1	8BUI	1	2BUI

```
-- Return bands 2 and 3. Using array cast syntax
SELECT ST_NumBands(ST_Band(rast, '{2,3}'::int[])) As num_bands
      FROM dummy_rast WHERE rid=2;

num_bands
-----
2

-- Return bands 2 and 3. Use array to define bands
SELECT ST_NumBands(ST_Band(rast, ARRAY[2,3])) As num_bands
      FROM dummy_rast
WHERE rid=2;
```



```
--Make a new raster with 2nd band of original and 1st band repeated twice,
and another with just the third band
SELECT rast, ST_Band(rast, ARRAY[2,1,1]) As dupe_band,
       ST_Band(rast, 3) As sing_band
FROM samples.than_chunked
WHERE rid=35;
```


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.

示例

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 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	

函数

ST_MakeEmptyRaster

11.3.6 ST_MakeEmptyRaster

ST_MakeEmptyRaster — 创建空栅格 (列 & 行), 列 X 行, 比例尺, 偏斜 (scalex, scaley, skewx & skewy) 和 SRID (SRID) 的栅格。 列, 行, 比例尺, 偏斜 SRID 是可选的。 SRID 未指定, 默认为 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);

返回

返回栅格 (栅格 & 栅格), 栅格 (栅格) 返回栅格 X(upperleftx) 栅格 Y(upperlefty), 栅格, 栅格 (scalex, scaley, skewx & skewy) 返回栅格 (SRID) 返回栅格 (栅格) 返回栅格。

返回栅格 (pixelsize) 返回栅格。 scalex 返回栅格, scaley 返回栅格。 skewx & skewy 0 返回栅格。

返回栅格, 返回栅格 (栅格) 返回栅格。

栅格 SRID 返回栅格 0 栅格。 返回栅格。 返回栅格 **ST_AddBand** 栅格, 返回栅格 **ST_SetValue** 栅格。

返回

```
INSERT INTO dummy_rast(rid,rast)
VALUES(3, ST_MakeEmptyRaster( 100, 100, 0.0005, 0.0005, 1, 1, 0, 0, 4326) );
```

```
--use an existing raster as template for new raster
INSERT INTO dummy_rast(rid,rast)
SELECT 4, ST_MakeEmptyRaster(rast)
FROM dummy_rast WHERE rid = 3;
```

```
-- output meta data of rasters we just added
SELECT rid, (md).*
FROM (SELECT rid, ST_MetaData(rast) As md
      FROM dummy_rast
      WHERE rid IN(3,4)) As foo;
```

```
-- output --
```

rid	upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	←
numbands										
3	0.0005	0.0005	100	100	1	1	0	0	4326	←
4	0.0005	0.0005	100	100	1	1	0	0	4326	←

返回

ST_AddBand, **ST_MetaData**, **ST_ScaleX**, **ST_ScaleY**, **ST_SetValue**, **ST_SkewX**, , **ST_SkewY**

11.3.7 ST_Tile

ST_Tile — 返回栅格。

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 padwithnodata=FALSE, double precision nodataval=NULL);
 setof raster **ST_Tile**(raster rast, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);

返回

返回栅格集合，每个栅格都是输入栅格的一个子集，其宽度和高度由参数指定。

padwithnodata = FALSE 时，返回的栅格集合中的每个栅格都是输入栅格的一个子集，其宽度和高度由参数指定。padwithnodata = TRUE 时，返回的栅格集合中的每个栅格都是输入栅格的一个子集，其宽度和高度由参数指定，并且每个栅格的边缘都填充了 NODATA 值 (padding) 指定的值。如果指定的 NODATA 值 (padding) 指定的值与输入栅格的 NODATA 值不同，则返回的栅格集合中的每个栅格都是输入栅格的一个子集，其宽度和高度由参数指定，并且每个栅格的边缘都填充了 NODATA 值指定的值。



Note

返回的栅格集合中的每个栅格都是输入栅格的一个子集，其宽度和高度由参数指定，并且每个栅格的边缘都填充了 NODATA 值指定的值。

2.1.0 版本中的变化

返回

```
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)
```

☐☐

[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');

☐☐

`sfx` (`sfx`) `sfy` (`sfy`) `tw` (`tw`) `th` (`th`) 指定栅格的宽度和高度。 `tab` (`col`) 指定栅格的名称。 `ext` 指定栅格的几何类型。

指定重采样方法：'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', 或 'Lanczos'。 默认值为 'NearestNeighbor'。 有关更多选项，请参阅 [GDAL Warp resampling methods](#)。

2.2.0 版本引入。

☐☐

[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);

☐☐

`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 的整数。 如果 `srid` 为 NULL，则使用 GDAL 数据集的 SRID。 如果 `srid` 为 NULL，则使用 GDAL 数据集的 SRID。

2.1.0 版本引入。

例

```
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)

例

ST_AsGDALRaster

11.4 访问器 (accessor)

11.4.1 ST_GeoReference

ST_GeoReference — 返回 (world) 坐标参考系 (CRS) 的 GDAL 或 ESRI 字符串。
返回 GDAL 字符串。

Synopsis

text **ST_GeoReference**(raster rast, text format=GDAL);

例

例 1 返回 GDAL 字符串, 例 2 (carriage) 返回 ESRI 字符串。
返回 GDAL 字符串。返回 'GDAL' 或 'ESRI' 字符串。

返回:

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 — 检查空栅格 (width = 0, height = 0) 是否有效。返回布尔值。

Synopsis

boolean **ST_IsEmpty**(raster rast);

函数

检查空栅格 (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))
st_isempty |
-----+
f          |

SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0))
st_isempty |
-----+
t          |
```

函数

ST_HasNoBand

11.4.4 ST_MemSize

ST_MemSize — 返回栅格在内存中的大小 (字节)。

Synopsis

integer **ST_MemSize**(raster rast);



11.4.6 ST_NumBands

Synopsis



11.4.7 ST_PixelHeight

Synopsis

XXXXXXXXXXXXXXXXXXXXXXXX ST PixelWidth XXXXXXXX.

图例: 栅格元数据

```
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

图例: 0 度偏斜栅格元数据

```
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

图例

[ST_PixelWidth](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SkewX](#), [ST_SkewY](#)

11.4.8 ST_PixelWidth

ST_PixelWidth — 返回栅格的像素宽度。

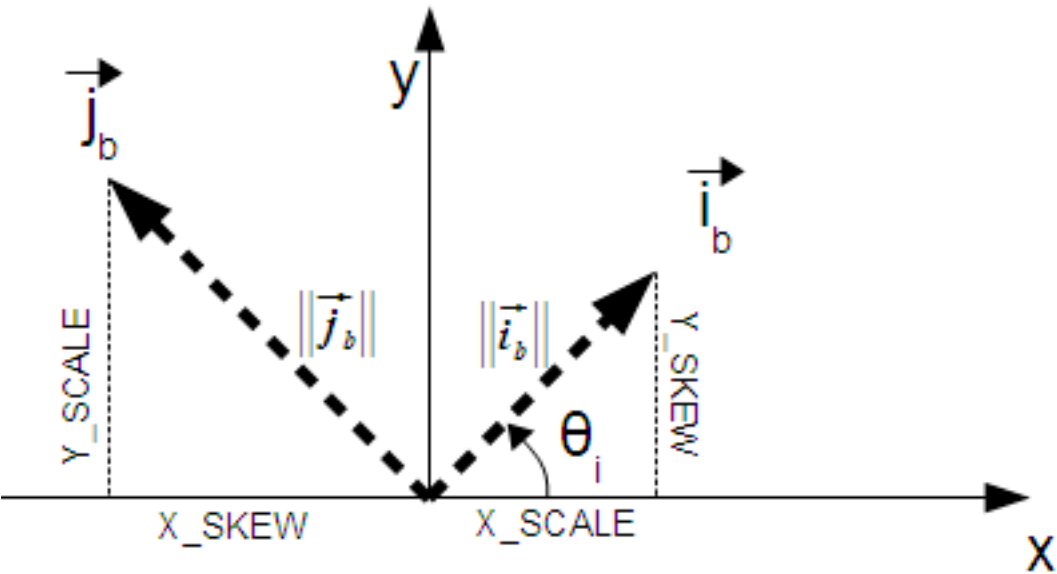
Synopsis

double precision **ST_PixelWidth**(raster rast);

图例

返回栅格的像素宽度。对于非正方形栅格，返回的像素宽度与像素高度不同。对于正方形栅格，返回的像素宽度与像素高度相同。

图例:



i
 j

例: 设置缩放和倾斜

```
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

例: 0 设置缩放和倾斜

```
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

例

ST_PixelHeight, ST_ScaleX, ST_ScaleY, ST_SkewX, ST_SkewY

11.4.9 ST_ScaleX

ST_ScaleX — 将栅格的 X 轴像素大小缩放为指定的值。

Synopsis

float8 **ST_ScaleX**(raster rast);

返回

返回栅格的 X 轴像素大小。如果指定的值不为 1，则返回的像素大小将乘以指定的缩放因子。
版本: 2.0.0 在 WKTRaster 中通过 ST_PixelSizeX 引入。

示例

```
SELECT rid, ST_ScaleX(rast) As rastpixwidth
FROM dummy_rast;
```

rid	rastpixwidth
1	2
2	0.05

注意

ST_Width

11.4.10 ST_ScaleY

ST_ScaleY — 将栅格的 Y 轴像素大小缩放为指定的值。

Synopsis

float8 **ST_ScaleY**(raster rast);

返回

返回栅格的 Y 轴像素大小。如果指定的值不为 1，则返回的像素大小将乘以指定的缩放因子。
版本: 2.0.0 在 WKTRaster 中通过 ST_PixelSizeY 引入。

示例

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;
```

rid	rastpixheight
1	3
2	-0.05

图例

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);

图例

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，则返回的坐标值将包含畸变。对于非倾斜栅格，返回的坐标值将是精确的。对于倾斜栅格，返回的坐标值将是近似的。对于倾斜栅格，返回的坐标值将是近似的。对于倾斜栅格，返回的坐标值将是近似的。

2.1.0 版本引入。

图例

```
-- 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

☐☐

[ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SetSkew](#)

11.4.12 ST_RasterToWorldCoordX

`ST_RasterToWorldCoordX` — 返回栅格 `rast` 中列 `xcolumn` 的 X 坐标。列索引从 1 开始。

Synopsis

```
float8 ST_RasterToWorldCoordX(raster rast, integer xcolumn);
float8 ST_RasterToWorldCoordX(raster rast, integer xcolumn, integer yrow);
```

☐☐

返回栅格 `rast` 中列 `xcolumn` 的 X 坐标。列索引从 1 开始。如果提供了行索引 `yrow`，则返回该像素的 X 坐标。



Note

如果栅格是倾斜的，X 坐标将不是整数。在这种情况下，`ST_ScaleX`, `ST_SkewX`, 和 `ST_SetSkew` 函数将返回 X 坐标的小数部分。

☐☐☐☐: 2.1.0 版本中 `ST_Raster2WorldCoordX` 函数被弃用。

☐☐

```
-- 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

¶

[ST_ScaleX](#), [ST_RasterToWorldCoordY](#), [ST_SetSkew](#), [ST_SkewX](#)

11.4.13 ST_RasterToWorldCoordY

`ST_RasterToWorldCoordY` — 返回栅格中指定行 Y 的坐标。返回类型是 `float8`。

Synopsis

```
float8 ST_RasterToWorldCoordY(raster rast, integer yrow);
float8 ST_RasterToWorldCoordY(raster rast, integer xcolumn, integer yrow);
```

¶

`ST_RasterToWorldCoordY` 返回栅格中指定行 Y 的坐标。返回类型是 `float8`。如果指定了 `xcolumn`，则返回该列的 X 坐标。如果未指定 `xcolumn`，则返回该行的 Y 坐标。如果指定了 `yrow`，则返回该行的 Y 坐标。如果未指定 `yrow`，则返回该行的 Y 坐标。



Note

如果指定了 `xcolumn`，则返回该列的 X 坐标。如果未指定 `xcolumn`，则返回该行的 Y 坐标。如果指定了 `yrow`，则返回该行的 Y 坐标。如果未指定 `yrow`，则返回该行的 Y 坐标。

注意: 2.1.0 版本中 `ST_Raster2WorldCoordY` 已被弃用。

¶

```
-- 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

返回

[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

返回栅格的旋转角度。如果栅格没有旋转，则返回 0。如果栅格是 NaN，则返回 NaN。如果栅格是空集，则返回 0。

返回

```
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

返回

[ST_SetRotation](#), [ST_SetScale](#), [ST_SetSkew](#)

11.4.15 ST_SkewX

`ST_SkewX` — 返回栅格的 X 轴偏斜角度 (skew)。

Synopsis

`float8 ST_SkewX(raster rast);`

返回

返回栅格的 X 轴偏斜角度 (skew)。如果栅格没有偏斜，则返回 0。如果栅格是 NaN，则返回 NaN。如果栅格是空集，则返回 0。

例

```
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

例

ST_GeoReference, ST_SkewY, ST_SetSkew

11.4.16 ST_SkewY

ST_SkewY — 浮点 Y 偏斜 (每像素) 值。

Synopsis

```
float8 ST_SkewY(raster rast);
```

例

浮点 Y 偏斜 (每像素) 值。 返回浮点值。

例

```
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



11.4.17 ST_SRID

Synopsis



Note!

PostGIS 2.0 1000000, 1000000000000/10000 SRID 1000000 -1 0 1000000000000.



11.4.18 ST_Summary

Synopsis

```
text ST_Summary(raster rast);
```


返回

[ST_UpperLeftY](#), [ST_GeoReference](#), [Box3D](#)

11.4.20 ST_UpperLeftY

`ST_UpperLeftY` — 返回栅格的 Y 坐标。

Synopsis

`float8 ST_UpperLeftY(raster rast);`

返回

返回栅格的 Y 坐标。

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 — Returns the X, Y (longitude, latitude) coordinates of the pixel at the given row and column of the raster.

Synopsis

record **ST_WorldToRasterCoord**(raster rast, geometry pt);
record **ST_WorldToRasterCoord**(raster rast, double precision longitude, double precision latitude);

Returns the X, Y (longitude, latitude) coordinates of the pixel at the given row and column of the raster. The X, Y coordinates are returned as a record. The X, Y coordinates are returned as a record. The X, Y coordinates are returned as a record.

2.1.0 Returns the X, Y coordinates.

```
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 — XXXXXX (pt) $\text{XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}$
X, Y XXXXXX (xw, yw) 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);
```



点 (pt) 的 X, Y 坐标 (xw, yw) 为。点 (xw, yw) 的 xw 为 yw 的。

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 — XXXXXX (pt) $\text{XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}$
X, Y XXXXX (xw, yw) 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);
```

注意

`ST_WorldToRasterCoordY` (pt) 返回栅格 `rast` 中 X, Y 坐标 (xw, yw) 的栅格坐标。栅格坐标 (xw, yw) 是栅格 `rast` 的 `xw` 和 `yw` 坐标的整数部分。栅格坐标 `xw` 和 `yw` 是栅格 `rast` 的 `xw` 和 `yw` 坐标的浮点部分。

版本: 2.1.0 添加 `ST_WorldToRasterCoordY` 函数。

注意

```
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
       ptcoord
FROM dummy_rast;
```

rid	ycoord	ycoord_xwyw	ptcoord
1	7	7	7
2	115864471	115864471	115864471

注意

[ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SRID](#)

11.5 栅格元数据

11.5.1 ST_BandMetaData

`ST_BandMetaData` — 返回栅格带的元数据。返回 1 个元数据记录。

Synopsis

- (1) record `ST_BandMetaData`(raster rast, integer band=1);
- (2) record `ST_BandMetaData`(raster rast, integer[] band);

注意

Returns basic meta data about a raster band. Columns returned: pixeltype, nodatavalue, isoutdb, path, outdbbandnum, filesize, filetimestamp.



Note

返回的元数据记录包含以下列。



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 到栅格的带数。

Synopsis

double precision **ST_BandNoDataValue**(raster rast, integer bandnum=1);

返回

指定 NODATA 值的栅格带的值。

示例

```
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 值的栅格带的布尔值。

Synopsis

boolean **ST_BandIsNoData**(raster rast, integer band, boolean forceChecking=true);
boolean **ST_BandIsNoData**(raster rast, boolean forceChecking=true);

返回

指定 NODATA 值的栅格带的布尔值。返回值的范围是 1 到栅格的带数。如果返回 TRUE，则表示指定的带是 NODATA 值。如果返回 FALSE，则表示指定的带不是 NODATA 值。

2.0.0 版本引入。

Note



在 2.0.0 版本中，`ST_BandIsNoData` 函数在 `forceChecking` 参数为 TRUE 时，会检查指定的带是否是 NODATA 值。如果返回 TRUE，则表示指定的带是 NODATA 值。如果返回 FALSE，则表示指定的带不是 NODATA 值。在 2.0.0 版本之前，`ST_BandIsNoData` 函数总是返回 TRUE。从 2.0.0 版本开始，`ST_SetBandIsNoData` 函数可以用于设置指定的带为 NODATA 值。

￼￼

```

-- 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 of a band stored in file system. `bandnum` defaults to 1.

Synopsis

text **ST_BandPath**(raster rast, integer bandnum=1);

`ST_BandPath`

Returns the path of a band stored in file system. DB access is required.

`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

返回

```
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 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 返回的字符串格式如下：

- 1BB - 1 字节
- 2BUI - 2 字节的无符号整数
- 4BUI - 4 字节的无符号整数
- 8BSI - 8 字节的有符号整数
- 8BUI - 8 字节的无符号整数
- 16BSI - 16 字节的有符号整数
- 16BUI - 16 字节的无符号整数

- 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
```

ST_BandPixelType

ST HasNoBand — $\exists x_1 \dots \exists x_{n-1} \exists y_1 \dots \exists y_n. \text{HasNoBand}(x_1, \dots, x_{n-1}, y_1, \dots, y_n) \wedge 1 \leq x_i \leq n$.

```
boolean ST_HasNoBand(raster rast, integer bandnum=1);
```

[illegible]

```
SELECT rid, ST_HasNoBand(rast) As hb1, ST_HasNoBand(rast,2) as hb2,
ST_HasNoBand(rast,4) as hb4, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

rid	hb1	hb2	hb4	numbands
1	t	t	t	0
2	f	f	t	3

ST NumBands

11.6.1 ST_PixelAsPolygon

ST_PixelAsPolygon — .

```
geometry ST_PixelAsPolygon(raster rast, integer columnx, integer rowy);
```

2.0.0

例

```
-- get raster pixel polygon
SELECT i,j, ST_AsText(ST_PixelAsPolygon(foo.rast, i,j)) As b1pgeom
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	b1pgeom
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 — ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

Synopsis

geometry **ST_PixelAsPoint**(raster rast, integer columnx, integer rowy);

¶¶

¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

2.1.0 ¶¶¶¶¶¶¶¶¶¶¶¶.

¶¶

```
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. The value is returned as a double precision floating point number.

Synopsis

setof record **ST_PixelAsPoints**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Notes

The `x`, `y` coordinates are returned as integers. The value is returned as a double precision floating point number.

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 Returns the `x`, `y` coordinates and the value of the pixel at the given location.

2.1.1 `exclude_nodata_value` boolean flag.

Notes

```
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)

返回

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#)

11.6.5 ST_PixelAsCentroid

`ST_PixelAsCentroid` — 返回指定栅格像素 (x, y) 的几何体。

Synopsis

geometry **ST_PixelAsCentroid**(raster rast, integer x, integer y);

返回

返回指定栅格像素 (x, y) 的几何体。

版本: 2.1.0 引入 C 语言实现。

2.1.0 版本引入。

返回

```
SELECT ST_AsText(ST_PixelAsCentroid(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;
```

```
st_astext
-----
POINT(1.5 2)
```

返回

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroids](#)

11.6.6 ST_PixelAsCentroids

`ST_PixelAsCentroids` — 返回指定栅格像素 (x, y) 的几何体 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 columnx, rowy 处的值, 如果排除 no_data 值。如果 columnx 或 rowy 为 1, 则返回 1。exclude_nodata_value 为 true 时, no_data 值将被排除。exclude_nodata_value 为 false 时, no_data 值将被返回。

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

columnx, rowy 处的值, 如果排除 no_data 值。如果 columnx 或 rowy 为 1, 则返回 1。

exclude_nodata_value 为 true 时, no_data 值将被排除。exclude_nodata_value 为 false 时, no_data 值将被返回。

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: exclude_nodata_value 为 true 时, no_data 值将被排除。

2.0.0 版本: exclude_nodata_value 为 true 时, no_data 值将被排除。

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 5793243.7,3427927.95  ←
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)))
```



```
        ),
        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)
```


图例

ST_Value, ST_SetSRID

11.6.11 ST_Neighborhood

ST_Neighborhood — columnx 列 rowy, 返回指定栅格中指定列和行周围的像素值。如果指定了 NODATA 值，则返回 2 个像素值。

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);
```

图例

columnx 列 rowy, 返回指定栅格中指定列和行周围的像素值。如果指定了 NODATA 值，则返回 2 个像素值。distanceX 列 distanceY 列。X 列 Y 列。如果指定了 NODATA 值，则返回 2 个像素值。2 个像素值。columnx 列 rowy 列。

如果指定了 1 个像素值，则返回 bandnum 列 1 个像素值。exclude_nodata_value 列。如果指定了 nodata 值，则返回 2 个像素值。exclude_nodata_value 列。如果指定了 nodata 值，则返回 2 个像素值。



Note

返回 2 个像素值。2 * (distanceX|distanceY) + 1 列。distanceX 列 distanceY 列 1 列，返回 3x3 个像素值。



Note

ST_Min4ma, ST_Sum4ma, ST_Mean4ma 返回指定栅格中指定列和行周围的像素值。2 个像素值。

2.1.0 新增功能。

图例

```
-- 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[],

```




11.6.13 ST_SetValues

Synopsis



2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.

测试: 测试 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...

```

<pre> + - + - + - + 1 1 1 + - + - + - + 1 1 1 + - + - + - + 1 1 1 + - + - + - + </pre>	=>	<pre> + - + - + - + 9 9 9 + - + - + - + 1 9 + - + - + - + 9 9 9 + - + - + - + </pre>

```

*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1,
            ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
            ARRAY[[false], [true]]::boolean[][]
        )
    ) AS poly
) foo
ORDER BY 1, 2;

```

x	y	val
1	1	9
1	2	9
1	3	9
2	1	9
2	2	
2	3	9
3	1	9
3	2	9
3	3	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

测试: 测试 2

```

/*
The ST_SetValues() does the following...

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 1 | 1 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      => | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
    SELECT
        ST_PixelAsPolygons(
            ST_SetValues(
                ST_AddBand(
                    ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 1, 1, ARRAY[[-1, -1, -1], [-1, 9, 9], [-1, 9, 9]]::double precision[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

测试: 测试 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, ← 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, ← 2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")
1	3	(1,"{{3,3,3,3,3},{3,NULL,NULL,NULL,NULL},{3,NULL,NULL,NULL,NULL},{3,NULL,NULL, ← NULL,NULL},{NULL,NULL,NULL,NULL,NULL}}")
1	4	(1,"{{4,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL, ← NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,4}}")

(4 rows)

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},{ ← NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")

(1 row)

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 = 2
      AND t3.gid = 1
ORDER BY t1.rid, t2.gid, t3.gid;
```

rid	gid	gid	st_dumpvalues
1	2	1	(1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,1,2,NULL},{
			NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")

(1 row)

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 — 返回 raster 中 columnx, rowy 位置的像素值。

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
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 格式为：

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 个参数的字符串。'GDAL' 或 'ESRI' 格式。GDAL 格式为 6 个参数，ESRI 格式为 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 版本中引入。在 2.1.0 之前，该函数在 2.1.0 版本中引入。

更新：2.1.0 版本中 `ST_SetGeoReference(raster, double precision, ...)` 函数。

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 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 — 将栅格旋转指定的角度。

Synopsis

raster **ST_SetRotation**(raster rast, float8 rotation);

☐☐

将栅格旋转指定的角度。返回的栅格具有与输入栅格相同的元数据。旋转角度以度为单位。

☐☐

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 和 Y 坐标乘以指定的标度因子。返回具有相同几何类型的栅格。

Synopsis

```
raster ST_SetScale(raster rast, float8 xy);
raster ST_SetScale(raster rast, float8 x, float8 y);
```

参数

X 和 Y 坐标乘以指定的标度因子。返回具有相同几何类型的栅格。如果指定了 X 和 Y 坐标，则分别对 X 和 Y 坐标进行缩放。



Note

ST_SetScale 与 ST_Rescale 类似，但 ST_Rescale 会重新采样栅格。ST_SetScale 不会重新采样栅格，而是返回具有相同几何类型的栅格。

版本: 2.0.0 在 WKTRaster 中，ST_SetPixelSize 与 ST_SetScale 类似。2.0.0 版本中，ST_SetPixelSize 与 ST_SetScale 类似。

示例

```
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)

返回

ST_ScaleX, ST_ScaleY, Box3D

11.7.4 ST_SetSkew

`ST_SetSkew` — Set the skew of a 2D geometry. `ST_SetSkew(geometry, float8 skewx, float8 skewy)`. The skew is defined as the angle of the X and Y axes relative to the horizontal.

Synopsis

```
raster ST_SetSkew(raster rast, float8 skewxy);
raster ST_SetSkew(raster rast, float8 skewx, float8 skewy);
```

Examples

Set the skew of a 2D geometry. The skew is defined as the angle of the X and Y axes relative to the horizontal. The skew is set to 1 for the X axis and 2 for the Y axis.

Example 1

```
-- 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 : 2.0000000000 : 1.0000000000 : 3.0000000000 : 0.5000000000 : 0.5000000000

Example 2 set both to same number:

```
UPDATE dummy_rast SET rast = ST_SetSkew(rast,0) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

rid	skewx	skewy	georef
1	0	0	2.0000000000 : 0.0000000000 : 0.0000000000 : 3.0000000000 : 0.5000000000 : 0.5000000000

See also

[ST_GeoReference](#), [ST_SetGeoReference](#), [ST_SkewX](#), [ST_SkewY](#)

11.7.5 ST_SetSRID

ST_SetSRID — 将 SRID 与 spatial_ref_sys 表中的 SRID 名称关联。

Synopsis

raster **ST_SetSRID**(raster rast, integer srid);

返回

与 SRID 名称关联的 raster。



Note

此函数将 SRID 与 spatial_ref_sys 表中的 SRID 名称关联。如果 spatial_ref_sys 表中不存在该 SRID，则函数将返回 NULL。如果 spatial_ref_sys 表中存在该 SRID，则函数将返回 raster。

返回

Section 4.5, [ST_SRID](#)

11.7.6 ST_SetUpperLeft

ST_SetUpperLeft — 设置栅格的左上角像素的投影 X 和 Y 坐标。

Synopsis

raster **ST_SetUpperLeft**(raster rast, double precision x, double precision y);

返回

设置栅格的左上角像素的投影 X 和 Y 坐标的 raster。

返回

```
SELECT ST_SetUpperLeft(rast, -71.01, 42.37)
FROM dummy_rast
WHERE rid = 2;
```

返回

[ST_UpperLeftX](#), [ST_UpperLeftY](#)

11.7.7 ST_Resample

ST_Resample — 对栅格进行重采样，指定新的宽度和高度，以及可选的网格大小、网格大小、倾斜度和倾斜度。返回重采样后的栅格。

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 preci-
sion 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);
```

参数

`width` 和 `height` (宽度和高度), `gridx` 和 `gridy` (网格大小和网格大小), `skewx` 和 `skewy` (倾斜度和倾斜度) 以及 `scalex`, `scaley`, `skewx` 和 `skewy` (缩放、缩放、倾斜度和倾斜度) 都是可选的。如果提供了 `ref`，则 `usescale` 默认为 `true`，否则默认为 `false`。SRID 是可选的。

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` 是可选的，默认为 0.125。



Note

有关 GDAL Warp 重采样方法的更多信息，请参阅 [GDAL Warp resampling methods](#)。

2.0.0 版本中首次引入。GDAL 1.6.1 版本中首次引入。

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 指定新像素大小，scaley 必须经常为负数以获得良好定向的栅格。ST_Resize 指定新像素大小。

maxerr 是变换近似的阈值（以像素单位）。如果未指定 maxerr，则使用默认值 0.125，这是 GDAL gdalwarp 工具使用的值。如果设置为零，则不进行近似。



Note

指定 GDAL Warp resampling methods 指定新像素大小。



Note

ST_Rescale 指定新像素大小，ST_SetScale 指定新像素大小。ST_SetScale 指定新像素大小（指定新像素大小）指定新像素大小。ST_Rescale 指定新像素大小，ST_SetScale 指定新像素大小。

2.0.0 指定新像素大小。GDAL 1.6.1 指定新像素大小。

Enhanced: 3.4.0 max and min resampling options added

指定新像素大小: 2.1.0 指定 SRID 指定新像素大小。

指定新像素大小

指定新像素大小 0.001 指定新像素大小 0.0015 指定新像素大小。

```
-- 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
```

指定新像素大小

ST_Resize, ST_Resample, ST_SetScale, ST_ScaleX, ST_ScaleY, ST_Transform

11.7.9 ST_Reskew

ST_Reskew — 对栅格 (栅格数据集) 进行重新采样。 NearestNeighbor(最近邻), Bilinear, Cubic, CubicSpline 和 Lanczos 是重新采样的方法。 NearestNeighbor 是默认方法。

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);
```

参数

skewx 和 **skewy** 是重新采样的角度。 NearestNeighbor(最近邻), Bilinear, Cubic, CubicSpline 和 Lanczos 是重新采样的方法。 NearestNeighbor 是默认方法。

skewx 和 **skewy** 是重新采样的角度。

重新采样的方法。

maxerr 是重新采样的最大误差 0.125 是默认值。



Note

重新采样的 [GDAL Warp resampling methods](#) 是重新采样的方法。



Note

ST_Reskew 是重新采样的方法。 **ST_SetSkew** 是重新采样的方法。 **ST_SetSkew** 是重新采样的方法。 (重新采样的方法) 是重新采样的方法。 **ST_Reskew** 是重新采样的方法。 **ST_SetSkew** 是重新采样的方法。

2.0.0 是重新采样的方法。 GDAL 1.6.1 是重新采样的方法。

重新采样的方法: 2.1.0 是重新采样的方法。 SRID 是重新采样的方法。

参数

重新采样的方法 0.0 是重新采样的方法 0.0015 是重新采样的方法。

```
-- 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
```


图例

图例显示了栅格数据在 snapping 前后的变化。

```
-- 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
```

图例

[ST_Resample](#), [ST_Rescale](#), [ST_UpperLeftX](#), [ST_UpperLeftY](#)

11.7.11 ST_Resize

ST_Resize — 将栅格数据重新调整大小/分辨率。

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);

图例

图例显示了栅格数据在重新调整大小/分辨率后的变化。图例显示了栅格数据在重新调整大小/分辨率后的变化。

NearestNeighbor(图例), Bilinear, Cubic, CubicSpline 图例 Lanczos 图例 NearestNeighbor 图例。

图例 1 图例显示了栅格数据在重新调整大小/分辨率后的变化。

图例 2 图例显示了栅格数据在重新调整大小/分辨率后的变化。

图例 3 图例显示了栅格数据在重新调整大小/分辨率后的变化。

2.1.0 图例显示了栅格数据在重新调整大小/分辨率后的变化。 GDAL 1.6.1 图例显示了栅格数据在重新调整大小/分辨率后的变化。

❏❏

```
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	↵
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)

❏❏

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);
```



0x00000000000000000000000000000000 (pixel warp) 0x000000000000000000000000
0x000000. 0x000000000000000000000000 NearestNeighbor 0x, maxerror 0x00000000
0x00000000 0.125 0x00.

GDAL Warp resampling methods: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', 'Lanczos'.

ST_Transform(*geom*) ST_SetSRID(*geom*, *SRID*). ST_Transform(*geom*, *SRID*)
 将 *geom* 从它的 SRID 转换为 *SRID*。如果 *geom* 是 *POINT* 类型，那么 *SRID* 是必需的。
 如果 *geom* 是 *POINT* 类型，那么 *SRID* 是必需的。如果 *geom* 是 *POINT* 类型，那么 *SRID* 是必需的。

00000000, 00 3 0 alignto 0000000000000000. 000000000000000000
 00 (SRID) 00000000, (ST SameAlignment = TRUE 000) 0000000000000000.

Note



If you find your transformation support is not working right, you may need to set the environment variable PROJSO to the .so or .dll projection library your PostGIS is using. This just needs to have the name of the file. So for example on windows, you would in Control Panel -> System -> Environment Variables add a system variable called PROJSO and set it to libproj.dll (if you are using proj 4.6.1). You'll have to restart your PostgreSQL service/daemon after this change.



Warning

When transforming a coverage of tiles, you almost always want to use a reference raster to insure same alignment and no gaps in your tiles as demonstrated in example: Variant 3.

2.0.0 11111111111111111111. GDAL 1.6.1 11111111111111111111.

OpenGL: 2.1.0 ST Transform(rast, alignto) ~~XXXXXXXXXX~~.



```
SELECT ST_Width(mass_stm) As w_before, ST_Width(wgs_84) As w_after,
       ST_Height(mass_stm) As h_before, ST_Height(wgs_84) As h_after
FROM
  ( SELECT rast As mass_stm, ST_Transform(rast,4326) As wgs_84
    , ST_Transform(rast,4326, 'Bilinear') AS wgs_84_bilin
    FROM aerials.o_2_boston
    WHERE ST_Intersects(rast,
                        ST_Transform(ST_MakeEnvelope(-71.128, 42.2392, -71.1277, 42.2397, 4326) ↵
                        ,26986) )
    LIMIT 1) As foo;
```

w_before	w_after	h_before	h_after
200	228	200	170

 栅格数据 (mass_stm)	 WGS84 栅格 (wgs_84) 栅格	 栅格 NN 栅格 Bilinear 栅格 栅格 WGS84 栅格 (wgs_84_bilin) 栅格

图 3

使用 `ST_Transform(raster, srid)` 或 `ST_Transform(raster, alignto)` 将栅格数据转换为指定的坐标系。

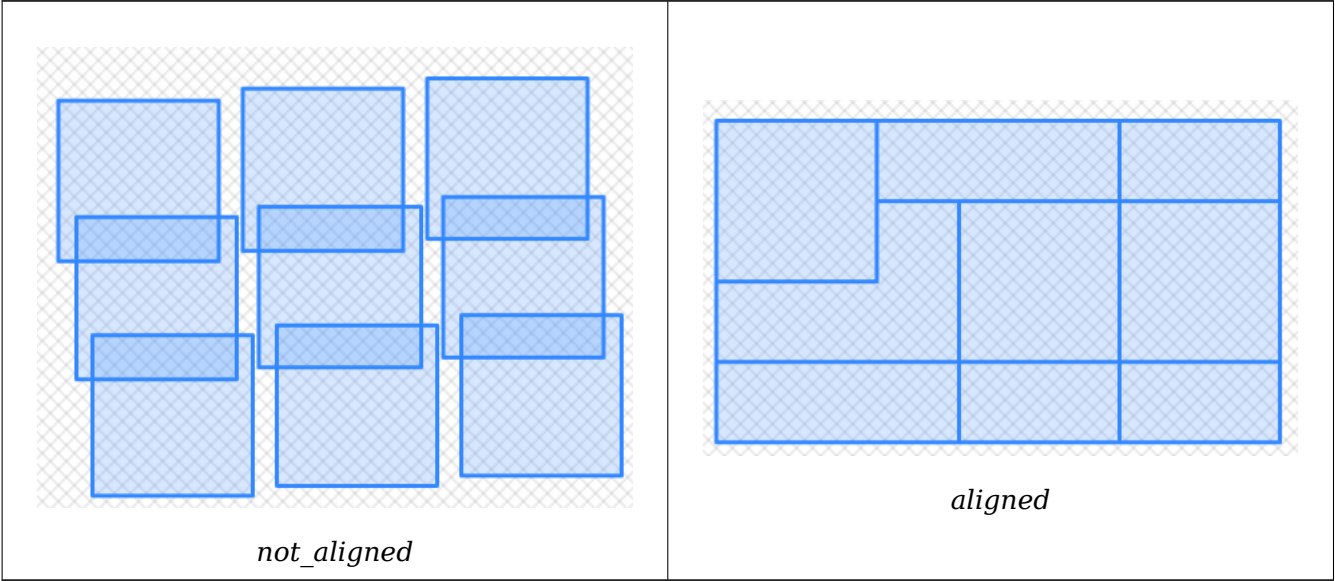
```
WITH foo AS (
  SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 600000, 100, -100, 0, 0,
    2163), 1, '16BUI', 1, 0) AS rast UNION ALL
  SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 600000, 100, -100, 0, 0, 2163),
    1, '16BUI', 2, 0) AS rast UNION ALL
  SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 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 简体中文

11.8.1 ST_SetBandNoDataValue

ST_SetBandNoDataValue — NODATA 简体中文. 简体中文. 1 简体中文. 简体中文 NODATA 简体中文, nodata value = NULL 简体中文.

Synopsis

```
raster ST_SetBandNoDataValue(raster rast, double precision nodatavalue);
raster ST_SetBandNoDataValue(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);
```


NODATA **ST Polygon, ST DumpAsPolygons,** **ST PixelAs...()**


```
-- 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;
```



ST BandNoDataValue, ST NumBands

11.8.2 ST_SetBandIsNoData

ST_SetBandIsNoData — `isnodata` `XXXXXXXXXX`.

Synopsis

```
raster ST_SetBandIsNoData(raster rast, integer band=1);
```

[illegible]

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

☒☒

```

-- 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
  ),
  ),
```



```
    ARRAY[1,3,2]::int[]
)
ORDER BY 1, 2;

query | bandnum | pixeltypes | nodatavalue | isoutdb | path | outdbbandnum
-----+-----+-----+-----+-----+-----+-----
1 | 1 | 8BUI | | t | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif | 1
1 | 2 | 8BUI | | t | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif | 2
1 | 3 | 8BUI | | t | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif | 3
2 | 1 | 8BUI | | t | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif | 1
2 | 2 | 8BUI | | t | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected2.tif | 1
2 | 3 | 8BUI | | t | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif | 3
```

[ST_BandMetaData](#), [ST_SetBandIndex](#)

11.8.4 ST_SetBandIndex

ST_SetBandIndex — Update the external band number of an out-db band

Synopsis

raster **ST_SetBandIndex**(raster rast, integer band, integer outdbindex, boolean force=false);

Updates an out-db band’s external band number. This does not touch the external raster file associated with the out-db band



Note If force is set to true, no tests are done to ensure compatibility (e.g. alignment, pixel support) between the external raster file and the PostGIS raster. This mode is intended for where bands are moved around in the external raster file.



Note Internally, this method replaces the PostGIS raster’s band at index band with a new band instead of updating the existing path information.

Availability: 2.5.0

❏❏

```
WITH foo AS (  
  SELECT  
    ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/  
    loader/Projected.tif', NULL::int[]) AS rast  
)  
SELECT  
  1 AS query,  
  *  
FROM ST_BandMetadata(  
  (SELECT rast FROM foo),  
  ARRAY[1,3,2]::int[]  
)  
UNION ALL  
SELECT  
  2,  
  *  
FROM ST_BandMetadata(  
  (  
    SELECT  
      ST_SetBandIndex(  
        rast,  
        2,  
        1  
      ) AS rast  
    FROM foo  
  ),  
  ARRAY[1,3,2]::int[]  
)  
ORDER BY 1, 2;
```

query	bandnum	pixeltype	nodatavalue	isoutdb	path
1	1	8BUI		t	/home/pele/devel/geo/postgis-git/ raster/test/regress/loader/Projected.tif
1	2	8BUI		t	/home/pele/devel/geo/postgis-git/ raster/test/regress/loader/Projected.tif
1	3	8BUI		t	/home/pele/devel/geo/postgis-git/ raster/test/regress/loader/Projected.tif
2	1	8BUI		t	/home/pele/devel/geo/postgis-git/ raster/test/regress/loader/Projected.tif
2	2	8BUI		t	/home/pele/devel/geo/postgis-git/ raster/test/regress/loader/Projected.tif
2	3	8BUI		t	/home/pele/devel/geo/postgis-git/ 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` 参数为 `true` 时，将 NODATA 值的像素排除在统计之外。如果设置为 `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);

返回

返回一个 64 位整数，表示在指定的 nband 中，非空值的数量。nband 默认为 1。

exclude_nodata_value 默认为 TRUE，表示在计算时忽略空值。如果为 FALSE，则包含空值。exclude_nodata_value 默认为 TRUE。

sample_percent 默认为 1.0，表示对每个像素进行 100% 的采样。sample_percent 的范围为 0 到 1。

2.2.0 版本引入。

示例

```
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)
```

参见

[ST_Count](#), [ST_SummaryStats](#), [ST_SetBandNoDataValue](#)

11.9.3 ST_Histogram

ST_Histogram — (bin; 返回指定栅格中每个bin的统计信息) 返回指定栅格中每个bin的统计信息。返回的统计信息包括min, max, count, percent。返回的统计信息包括nband 1 个。

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);

返回

返回的统计信息包括min, max, count, percent。返回的统计信息包括nband 1 个。



Note

如果栅格包含nodata值，则返回的统计信息将不包括这些值。如果exclude_nodata_value为true，则返回的统计信息将不包括这些值。

width width: 指定每个bin的宽度。如果width为NULL，则width为栅格的宽度。如果width为数组，则width为每个bin的宽度。

例如: 9 个bin, width [a, b, c] 为 [a, b, c, a, b, c, a, b, c] 个bin。

bins (breakout) 指定: 指定每个bin的统计信息。返回的统计信息包括min, max, count, percent。

right 指定: 指定每个bin的统计信息。X 指定每个bin的统计信息 [a, b] 为 (a, b] 个bin。

Changed: 3.1.0 Removed ST_Histogram(table_name, column_name) variant.

2.0.0 返回的统计信息包括min, max, count, percent。

例如: 返回的统计信息 - 返回 1, 2, 3 个bin的统计信息。

```
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

图 11.9.3: 表 2 的 ST_Histogram 函数结果 6 个桶。

```
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)

图 11.9.4

[ST_Count](#), [ST_SummaryStats](#), [ST_SummaryStatsAgg](#)

11.9.4 ST_Quantile

`ST_Quantile` — 返回指定 (population) 的指定 (quantile) 的百分位数。例如，`ST_Quantile` 25%, 50%, 75% 的百分位数。

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);

注意

ST_Quantile (population) ST_Quantile (quantile) 注意
 注意, 注意 25%, 50%, 75% 注意 (percentile) 注意.



Note

exclude_nodata_value 注意, NODATA 注意.

Changed: 3.1.0 Removed ST_Quantile(table_name, column_name) variant.

2.0.0 注意.

注意

```
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` — 返回指定栅格的统计信息，包括 count, sum, mean, stddev, min, max 等。返回结果是一个表，包含 1 个记录。

Synopsis

```
summarystats ST_SummaryStats(raster rast, boolean exclude_nodata_value);
summarystats ST_SummaryStats(raster rast, integer nband, boolean exclude_nodata_value);
```

☐☐

返回的统计信息包括 count, sum, mean, stddev, min, max 等。返回结果是一个表，包含 1 个记录。返回结果是一个表，包含 1 个记录。



Note

如果栅格包含 nodata 值，则必须指定 `exclude_nodata_value` 参数。如果未指定，则默认值为 true。



Note

在 2.2.0 版本中，ST_SummaryStats 函数不再支持 sample_percent 参数。在 2.0.0 版本中，sample_percent 参数默认为 1。

2.2.0 版本中，ST_SummaryStats(rastertable, rastercolumn, ...) 函数不再支持。请使用 **ST_SummaryStatsAgg** 函数。

2.0.0 版本中，ST_SummaryStats 函数不再支持。

SQL: 统计栅格数据

```
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

SQL: 统计栅格数据 (PostGIS 3.6.0alpha1)

PostGIS 3.6.0alpha1 版本中，ST_SummaryStats 函数不再支持。在 2.0.0 版本中，ST_SummaryStats 函数不再支持。在 2.2.0 版本中，ST_SummaryStats 函数不再支持。

```
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

例: 统计每个波段

```
-- 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

例

summarystats, ST_SummaryStatsAgg, ST_Count, ST_Clip

11.9.6 ST_SummaryStatsAgg

ST_SummaryStatsAgg — 聚合统计。返回 count, sum, mean, stddev, min, max 统计结果。聚合统计 1 个波段。

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);
```

例

聚合统计 count, sum, mean, stddev, min, max 统计结果 **summarystats** 聚合统计 nband 个波段。



Note

如果 `exclude_nodata_value` 为 `TRUE`，则统计结果将排除 `nodata` 值。如果 `exclude_nodata_value` 为 `FALSE`，则统计结果将包含 `nodata` 值。



Note

ST_SummaryStatsAgg 函数支持 sample_percent 参数，其值在 0 到 1 之间，表示对输入数据进行随机采样，并基于采样数据计算统计值。

2.2.0 新增功能。

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

ST_ValueCount — 返回指定栅格 (栅格) 中指定值的统计信息。返回 1 个值。返回 NODATA 值。返回 NODATA 值。返回 NODATA 值。

Synopsis

```
SETOF record ST_ValueCount(raster rast, integer nband=1, boolean exclude_nodata_value=true,
double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record ST_ValueCount(raster rast, integer nband, double precision[] searchvalues, double
precision roundto=0, double precision OUT value, integer OUT count);
SETOF record ST_ValueCount(raster rast, double precision[] searchvalues, double precision roundto=0,
double precision OUT value, integer OUT count);
bigint ST_ValueCount(raster rast, double precision searchvalue, double precision roundto=0);
bigint ST_ValueCount(raster rast, integer nband, boolean exclude_nodata_value, double precision
searchvalue, double precision roundto=0);
bigint ST_ValueCount(raster rast, integer nband, double precision searchvalue, double precision
roundto=0);
SETOF record ST_ValueCount(text rastertable, text rastercolumn, integer nband=1, boolean 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);
```

返回

返回 value, count 值。

nband 1 个值。searchvalues 值, 返回 NODATA 值。



Note

exclude_nodata_value 值, NODATA 值。

2.0.0 版本。

返回

```
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	



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(
    '0100000000000000000000000040000000000000008400000000000000000 ↵
      E03F00000000000000E03F0000000000000000000000000000000000A0000000A001400'
  )
)).* 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

```
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 — JPEG(Joint Photographic Exports Group) (AsJPEG) format. It is a lossy compression format. It uses 1 or 3 color channels. It is a 3 channel RGB format.

Synopsis

```
bytea ST_AsJPEG(raster rast, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer nband, integer quality);
bytea ST_AsJPEG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, integer quality);
```

描述

ST_AsJPEG(raster rast, text[] options=NULL) 使用 JPEG (Joint Photographic Experts Group) 格式将栅格数据导出为字节数组。ST_AsJPEG(raster rast, integer nband, integer quality) 使用 ST_AsGDALRaster 将栅格数据导出为字节数组。ST_AsJPEG(raster rast, integer nband, text[] options=NULL) 使用 ST_AsGDALRaster 将栅格数据导出为字节数组。ST_AsJPEG(raster rast, integer[] nbands, text[] options=NULL) 使用 ST_AsGDALRaster 将栅格数据导出为字节数组。ST_AsJPEG(raster rast, integer[] nbands, integer quality) 使用 ST_AsGDALRaster 将栅格数据导出为字节数组。

- **nband** - 指定要导出的波段数量。
- **nbands** - 指定要导出的波段列表 (JPEG 支持 3 个波段)。指定 RGB 波段。指定 ARRAY[3,2,1] 指定 3 个波段, 第 2 个波段, 第 1 个波段。
- **quality** - 1 到 100 之间的整数。指定 JPEG 的质量。
- **options** - JPEG 的 GDAL 选项 (ST_GDALDrivers 中的 JPEG 选项 create_options)。JPEG 选项, 指定 PROGRESSIVE ON/OFF 指定 75 到 0 到 100 指定 QUALITY 指定。指定 GDAL 选项。

2.0.0 版本。GDAL 1.6.0 版本。

用法:

```
-- output first 3 bands 75% quality
SELECT ST_AsJPEG(rast) As rastjpg
FROM dummy_rast WHERE rid=2;

-- output only first band as 90% quality
SELECT ST_AsJPEG(rast,1,90) As rastjpg
FROM dummy_rast WHERE rid=2;

-- output first 3 bands (but make band 2 Red, band 1 green, and band 3 blue, progressive and 90% quality
SELECT ST_AsJPEG(rast,ARRAY[2,1,3],ARRAY['QUALITY=90','PROGRESSIVE=ON']) As rastjpg
FROM dummy_rast WHERE rid=2;
```

描述

Section 10.3, ST_GDALDrivers, ST_AsGDALRaster, ST_AsPNG, ST_AsTIFF

11.11.5 ST_AsPNG

ST_AsPNG — 使用 PNG (Portable Network Graphics) 格式将栅格数据导出为字节数组 (PNG 格式) 将栅格数据导出为字节数组。ST_AsPNG(raster rast, integer nband, integer quality) 使用 ST_AsGDALRaster 将栅格数据导出为字节数组。ST_AsPNG(raster rast, integer nband, text[] options=NULL) 使用 ST_AsGDALRaster 将栅格数据导出为字节数组。ST_AsPNG(raster rast, integer[] nbands, text[] options=NULL) 使用 ST_AsGDALRaster 将栅格数据导出为字节数组。ST_AsPNG(raster rast, integer[] nbands, integer quality) 使用 ST_AsGDALRaster 将栅格数据导出为字节数组。

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);
```

返回

返回的 PNG (Portable Network Graphics) 格式的二进制数据。返回的字节数组是 **ST_AsGDALRaster** 返回的二进制数据。返回的字节数组，由 3 个字节组成。srid 返回的 SRID 值。返回的字节数组：

- nband - 返回的字节数组。
- nbands - 返回的字节数组 (PNG 格式 4 字节)。返回的 RGBA 格式。返回的 ARRAY[3,2,1] 3 个字节，由 2 个字节，由 1 个字节组成。
- compression - 1 到 9 的压缩级别。返回的字节数组。
- options - PNG 格式的二进制数据 GDAL 格式的二进制数据 (**ST_GDALDrivers** 返回 PNG 格式的二进制数据 create_options 返回的二进制数据)。PNG 格式，返回的 ZLEVEL(返回的二进制数据 - 返回的 6) 返回的，返回的 ARRAY['ZLEVEL=9'] 返回的。返回的 2 个字节组成。返回的 GDAL 格式的二进制数据。

2.0.0 返回的字节数组。GDAL 1.6.0 返回的字节数组。

返回

```
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;
```

返回

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=sameassource);
bytea ST_AsTIFF(raster rast, text compression="", integer srid=sameassource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text compression="", integer srid=sameassource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text[] options, integer srid=sameassource);
```


TIFF(Tagged Image File Format) 文件。 使用 **ST_AsGDALRaster** 函数。 使用 **ST_AsGDALRaster** 函数。 SRS 使用 **ST_AsGDALRaster** 函数。

- **nbands** - 图像的波段数 (PNG 最多 3 个波段). 如果是 RGB 图像. 通常是 ARRAY[3,2,1] 表示 3 个波段, 2 个波段, 1 个波段.
- **compression** - 压缩格式: JPEG90(可选), LZW, JPEG, DEFLATE9
- **options** - GTiff 的 GDAL 选项 (ST_GDALDrivers 的 GTiff 选项 create_options). 通常是 GDAL 的选项.
- **srid** - 空间参考系统 SRID 值.

2.0.0. GDAL 1.6.0.

☒☒: **JPEG** ☒☒ **90%**

```
SELECT ST_AsTIFF(rast, 'JPEG90') As rasttiff
FROM dummy rast WHERE rid=2;
```


ST GDALDrivers, ST AsGDALRaster, ST SRID

11.12 ☒☒☒☒☒☒☒

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;
```



数据库: 数据库

```
-- 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;
```



数据库

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 个波段。返回 1 个栅格。

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 个波段。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 个波段。

colormap 指定要使用的 8BUI 格式 colormap 格式。

colormap 格式:

- grayscale 指定 grayscale - 8BUI 格式 (shades of gray) 格式
- pseudocolor - 8BUI(RGBA) 格式 4 个波段，指定要使用的 8BUI 格式
- fire - 8BUI(RGBA) 格式 4 个波段，指定要使用的 8BUI 格式
- bluered - 8BUI(RGBA) 格式 4 个波段，指定要使用的 8BUI 格式

colormap 格式 (指定要使用的 8BUI 格式) 指定要使用的 8BUI 格式。指定要使用的 8BUI 格式 5 个波段。指定要使用的 8BUI 格式，指定要使用的 8BUI 格式 (RGBA) 格式 (指定要使用的 0 到 255 个波段)。指定要使用的 8BUI 格式/指定要使用的 0/100% 指定要使用的 8BUI 格式。指定要使用的 8BUI 格式，指定要使用的 8BUI 格式/指定要使用的 8BUI 格式。指定要使用的 8BUI 格式 NODATA 指定要使用的 8BUI 格式, nv, null 指定要使用的 8BUI 格式。指定要使用的 8BUI 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) 指定要使用的 gdaldem 格式。

method 指定要使用的 8BUI 格式:

- INTERPOLATE - 指定要使用的 8BUI 格式。
- EXACT - 指定要使用的 8BUI 格式。指定要使用的 8BUI 格式 0 0 0 0(RGBA) 指定要使用的 8BUI 格式。
- NEAREST - 指定要使用的 8BUI 格式。



Note

指定要使用的 ColorBrewer 格式。



Warning
ST_SetBandNoDataValue 函数在 NODATA 值上操作时，NODATA 值不会被修改。
ST_SetBandNoDataValue 函数在 NODATA 值上操作时，NODATA 值不会被修改。

2.1.0 版本更新。

更新

更新内容。

```
-- setup test raster table --
DROP TABLE IF EXISTS funky_shapes;
CREATE TABLE funky_shapes(rast raster);

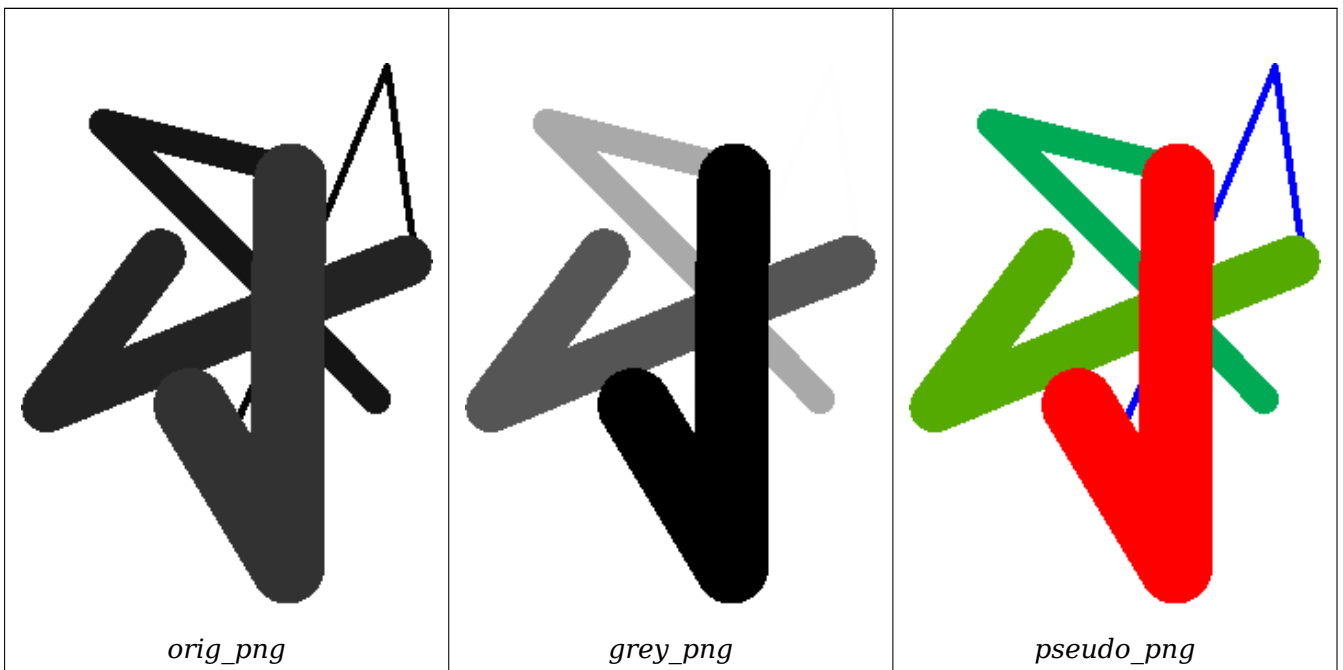
INSERT INTO funky_shapes(rast)
WITH ref AS (
  SELECT ST_MakeEmptyRaster( 200, 200, 0, 200, 1, -1, 0, 0) AS rast
)
SELECT
  ST_Union(rast)
FROM (
  SELECT
    ST_AsRaster(
      ST_Rotate(
        ST_Buffer(
          ST_GeomFromText('LINESTRING(0 2,50 50,150 150,125 50)'),
          i*2
        ),
        pi() * i * 0.125, ST_Point(50,50)
      ),
      ref.rast, '8BUI'::text, i * 5
    ) AS rast
  FROM ref
  CROSS JOIN generate_series(1, 10, 3) AS i
) AS shapes;
```

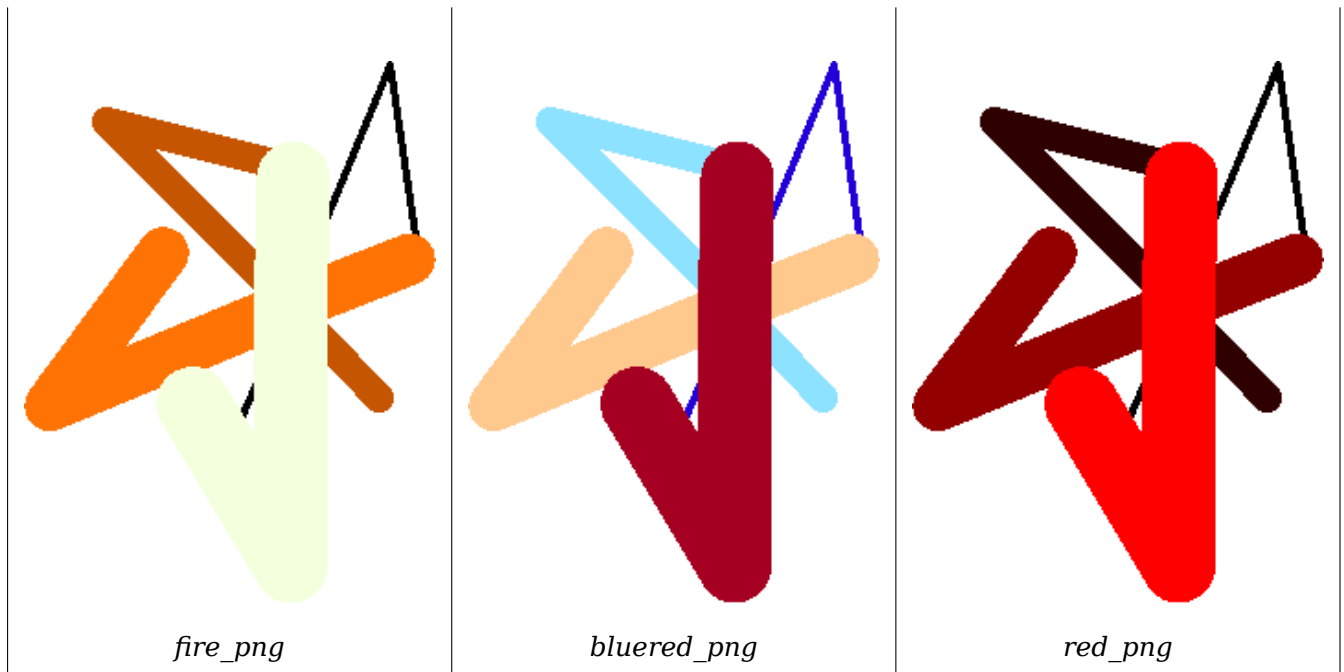
```
SELECT
  ST_NumBands(rast) As n_orig,
  ST_NumBands(ST_ColorMap(rast,1, 'greyscale')) As ngrey,
  ST_NumBands(ST_ColorMap(rast,1, 'pseudocolor')) As npseudo,
  ST_NumBands(ST_ColorMap(rast,1, 'fire')) As nfire,
  ST_NumBands(ST_ColorMap(rast,1, 'bluered')) As nbluered,
  ST_NumBands(ST_ColorMap(rast,1, '
100% 255  0  0
80% 160  0  0
50% 130  0  0
30%  30  0  0
20%  60  0  0
0%   0  0  0
nv 255 255 255
')) As nred
FROM funky_shapes;
```

n_orig	ngrey	npseudo	nfire	nbluered	nred
1	1	4	4	4	3

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 — $\{x \in \mathbb{R}^n : x \in S \text{ and } x \in T\}$, $\{x \in \mathbb{R}^n : x \in S \cap T\}$, $\{x \in \mathbb{R}^n : x \in S \text{ and } x \in T\}$.

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). geomval ST_Intersection(geometry, geometry) PostGIS NODATA WHERE ST_Intersect
```

```
geomval ST_Intersection(rast, geom).geom' rast.val' ST_Intersection(rast, geom).geom
ST_Intersection(rast, geom).geom
```

```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXXXXXXX ST_MapAlgebraExpr
XXXXXXXXXX 2 XXXXXXXXXXXXXXXXXXXX.

```

```

XXXXXXXXXXXXXXXXXXXXX.XXXXXXXXXXXXXXXXXXXXXX.returnband XXXXXXXX
XXXXXXXXXXXX' BAND1', 'BAND2' XXX' BOTH' XXXXXXXXXX. XXXXXXXX NODATA XXX
XXXXXXXXXXXXXXXXXXXXX NODATA XXXXXXXXX. X, NODATA XXXXXXXXXXXXXXXX
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
如果输入是 NODATA 值，则返回 NODATA。ST_MapAlgebraExpr 返回 NODATA。



Note
ST_Clip 返回与 ST_Clip 相同的几何类型。ST_Clip 返回与 ST_Clip 相同的几何类型。



Note
ST_Intersects 返回与 ST_Intersection 相同的几何类型。

PostGIS 2.0.0 版本中，ST_Intersects 返回与 ST_Intersection 相同的几何类型。

注意：如果输入是 NODATA 值，则返回 NODATA。

```
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	

注意：

geomval, ST_Intersects, ST_MapAlgebraExpr, ST_Clip, ST_AsText

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].

```

pixeltype pixeltype XXXXXX, XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX. pixeltype X NULL
XXXXXX, XXXXXX XXXXXXXXXXXXXXXXXXXX (XXXXXX: INTERSECTION,
UNION, FIRST, CUSTOM), XXXXXXXXXXXXXXXXXXXX (XXXXXX: SECOND, LAST) XX
XXXXXXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXXXXXXX. XXXX pixeltype XXXXXXXXXX.

```

```
ST_BandPixelType ST_BandPixelType, ST_BandPixelType, NULL
```

extenttype `INTERSECTION()`, `UNION`, `FIRST(1)`, `SECOND`, `LAST`, `CUSTOM`.

```
customextent extenttype CUSTOM XXX, XXXX customextent XXXXXXXXXXXX. XX 1 X 4
XXXXXXXXXXXXXXXXXX.
```

distance_x The distance in pixels from the reference cell in x direction. So width of resulting matrix would be $2 \times \text{distance}_x + 1$. If not specified only the reference cell is considered (neighborhood of 0).

distancey $\times \times \times \times \times \times$ Y $\times \times \times \times \times \times \times \times \times \times \times \times$. $\times \times \times \times \times \times \times \times \times \times$ 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) $\times \times \times \times \times \times$.

```
userargs callbackfunc [flags] variadic text [flags]. [flags]
    [callbackfunc flags], userargs [flags]
```


Note!

Note

(`VARIADIC`) `VARIADIC`, PostgreSQL `VARIADIC` `Query Language (SQL) Functions` "SQL Functions with Variable Numbers of Arguments" `VARIADIC`.

Note!

Note

```
x = x + y; callbackfunc(x);  
x = text[0];
```

§ 87(1)(b) § 87(2)(g)/§ 87(2)(b) § 87(2)(c) § 87(2)(d) § 87(2)(e) § 87(2)(f) § 87(2)(h) § 87(2)(i) § 87(2)(j) § 87(2)(k) § 87(2)(l) § 87(2)(m) § 87(2)(n) § 87(2)(o) § 87(2)(p) § 87(2)(q) § 87(2)(r) § 87(2)(s) § 87(2)(t) § 87(2)(u) § 87(2)(v) § 87(2)(w) § 87(2)(x) § 87(2)(y) § 87(2)(z) rastbandarg § 87(2)(aa) § 87(2)(ab). § 87(1)(b) § 87(2)(g) § 87(2)(b).

$\square\square 2 \square 3 \square\square\square\square\square\square 1 \square\square\square\square\square\square\square\square\square\square\square\square\square\square. \square\square 2 \square 3 \square\square\square\square\square\square\square\square\square\square.$

□□ 4 □□□□□□□□ 1 □□□□□□□□□□ 2 □□□□□□□□□□. □□ 4 □□□□□□□□□□.

2.2.0 `mask` `XXXXXXXXXX`.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.

图 1

图 1, 图 1

```
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
```

图 1, 图 1

```
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
```

图 1, 图 1

```
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
```

图 1. PostgreSQL 9.1 图 1.

```
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

```

PostgreSQL 9.0 简体中文版。

```

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;

```

测试: 测试 2 测试 3

测试测试 1 测试, 测试测试

```

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

```

测试测试 1 测试, 测试 1 测试

```

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

```

测试: 测试 4

测试测试 2 测试, 测试 2 测试

```

WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↵
        0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION ↵
        ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ↵
        0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)

```

```

SELECT
  ST_MapAlgebra(
    t1.rast, 2,
    t2.rast, 1,
    'sample_callbackfunc(double precision[], int[], text[])::regprocedure'
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
      AND t2.rid = 2

```

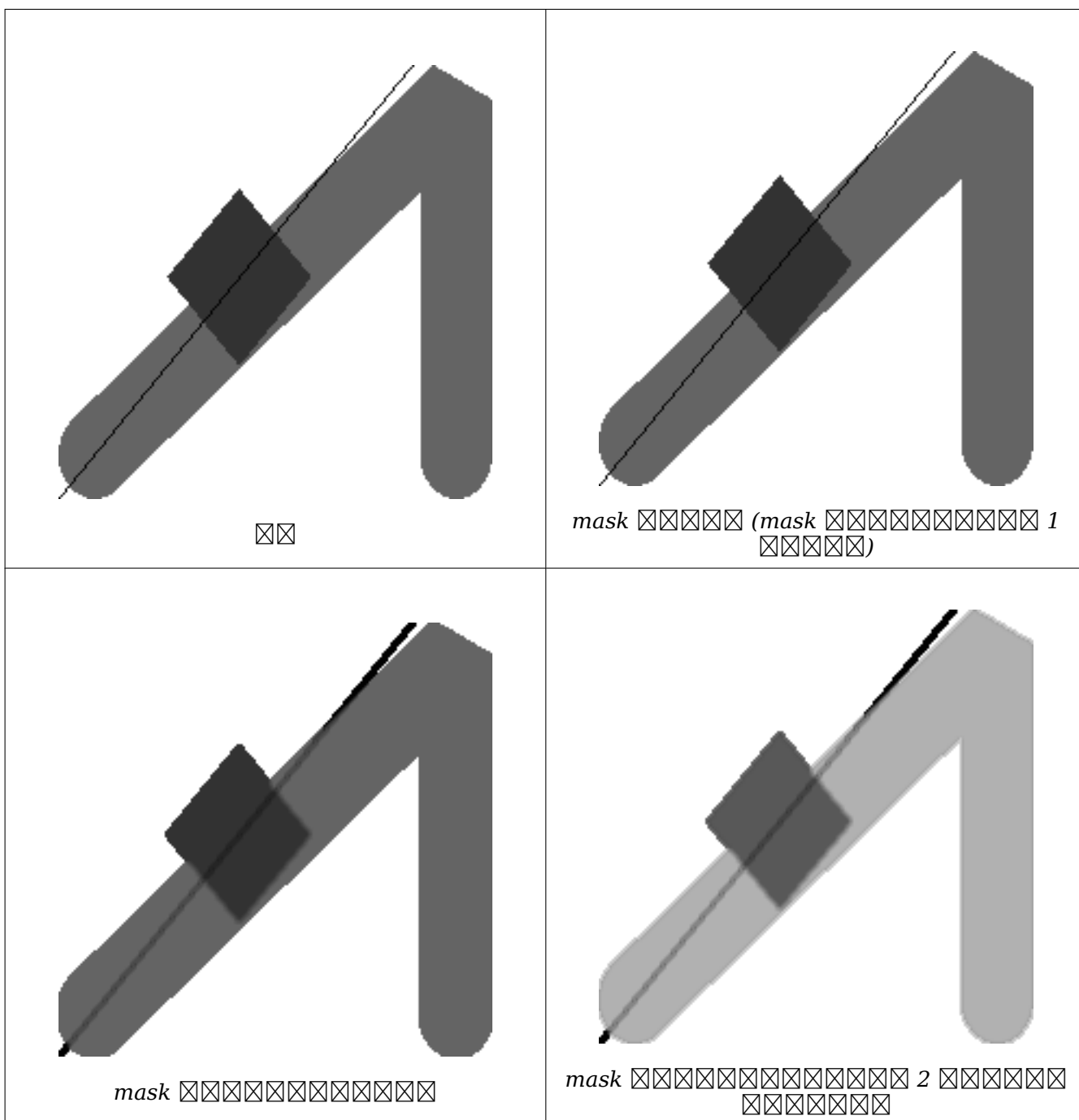
mask

```

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;

```



rastbandarg, ST_Union, ST_MapAlgebra (expression version)

11.12.6 ST_MapAlgebra (expression version)

ST_MapAlgebra (expression version) — $\sum_{i=1}^n \text{ST\_Area}(A_i) - \sum_{i=1}^n \text{ST\_Area}(B_i)$, $\sum_{i=1}^n \text{ST\_Area}(A_i) + \sum_{i=1}^n \text{ST\_Area}(B_i)$, $\sum_{i=1}^n \text{ST\_Area}(A_i) * \sum_{i=1}^n \text{ST\_Area}(B_i)$.

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

返回类型 - 返回类型 1 返回类型 2 是, 返回类型, 返回类型 SQL 返回类型 1 返回类型返回类型 1 返回类型返回类型返回类型。

2.1.0 返回类型返回类型。

参数: 参数 1, 2 (参数 1 是)

rast (rast) 返回 expression 返回类型 PostgreSQL 返回类型返回类型, 返回类型 1 返回类型返回类型返回类型。 **nband** 返回类型返回类型, 返回类型 1 返回类型返回类型。返回类型返回类型返回类型返回类型, 返回类型返回类型, 返回类型 1 返回类型返回类型。

pixeltype 返回类型, 返回类型返回类型返回类型返回类型返回类型。 **pixeltype** 是 NULL 返回类型, 返回类型返回类型返回类型 rast 返回类型返回类型返回类型返回类型。

• expression 返回类型返回类型。

1. [rast] - 返回类型返回类型
2. [rast.val] - 返回类型返回类型
3. [rast.x] - 返回类型 1-返回类型
4. [rast.y] - 返回类型 1-返回类型

参数: 参数 3, 4 (参数 2 是)

返回类型 rast1, (rast2) 返回 expression 返回类型 2 返回类型, 返回 PostgreSQL 返回类型返回类型返回类型, 返回类型 1 返回类型返回类型返回类型返回类型。 **band1**, **band2** 返回类型返回类型, 返回类型 1 返回类型返回类型。返回类型返回类型返回类型返回类型返回类型返回类型 (返回类型, 返回类型返回类型返回类型) 返回类型返回类型。 **extenttype** 返回类型返回类型返回类型返回类型返回类型。

expression 返回类型 2 返回类型 PostgreSQL 返回类型返回类型返回类型返回类型返回类型 PostgreSQL 返回类型/返回类型返回类型。 返回类型: (([rast1] + [rast2])/2.0)::integer

pixeltype 返回类型返回类型返回类型。返回类型 **ST_BandPixelType** 返回类型返回类型返回类型返回类型, 返回类型, NULL 返回类型返回类型。返回类型返回类型 NULL 返回类型返回类型, 返回类型返回类型返回类型返回类型返回类型。

extenttype 返回类型返回类型

1. INTERSECTION - 返回类型返回类型返回类型返回类型返回类型返回类型。返回类型返回类型。
2. UNION - 返回类型返回类型返回类型返回类型返回类型返回类型。
3. FIRST - 返回类型返回类型返回类型返回类型返回类型返回类型。

4. SECOND - 返回栅格值。

nodata1expr rast1 栅格 NODATA 值, rast2 栅格, 返回 rast2 栅格值。

nodata2expr rast2 栅格 NODATA 值, rast1 栅格, 返回 rast1 栅格值。

nodatanodataval 栅格 rast1 和 rast2 栅格 NODATA 值。

• expression, nodata1expr 和 nodata2expr 栅格。

1. [rast1] - rast1 栅格
2. [rast1.val] - rast1 栅格值
3. [rast1.x] - rast1 栅格 1-栅格
4. [rast1.y] - rast1 栅格 1-栅格
5. [rast2] - rast2 栅格
6. [rast2.val] - rast2 栅格值
7. [rast2.x] - rast2 栅格 1-栅格
8. [rast2.y] - rast2 栅格 1-栅格

图: 图 1 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;
```

图: 图 3 4

```
WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI'::text, 100, 0) AS rast
  UNION ALL
  SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI'::text, 300, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    t1.rast, 2,
    t2.rast, 1,
    '([rast2] + [rast1.val]) / 2'
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
AND t2.rid = 2;
```

图

rastbandarg, ST_Union, ST_MapAlgebra (callback function version)

11.12.7 ST_MapAlgebraExpr

ST_MapAlgebraExpr — 返回栅格 1 的表达式: 返回栅格表达式 PostgreSQL 表达式, 栅格 1 的表达式, 栅格 1 的表达式. 返回栅格表达式, 栅格 1 的表达式.

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);

栅格



Warning

ST_MapAlgebraExpr 2.1.0 版本中已弃用. 使用 **ST_MapAlgebra (expression version)** 代替.

expression (rast) 返回 expression 返回 PostgreSQL 表达式, 栅格 1 的表达式. **band** 返回栅格, 栅格 1 的表达式. 返回栅格表达式, 栅格 1 的表达式.

pixeltype 返回栅格, 返回栅格表达式. **pixeltype** 为 NULL 时, 返回栅格 rast 返回栅格表达式.

返回栅格表达式 [rast], 1-返回栅格表达式 [rast.x], 1-返回栅格表达式 [rast.y] 返回栅格表达式.

2.0.0 版本中已弃用.

栅格

返回栅格 2 返回栅格表达式 (modulo) 返回栅格表达式 1 返回栅格表达式.

```
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

將 NODATA 值 0 轉換為 2BUI 值, 1 轉換為 2BUI 值。

```
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		1
252		2
253		3
254		3

```
SELECT
    ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast
WHERE rid = 2;
```

b1pixtyp
2BUI

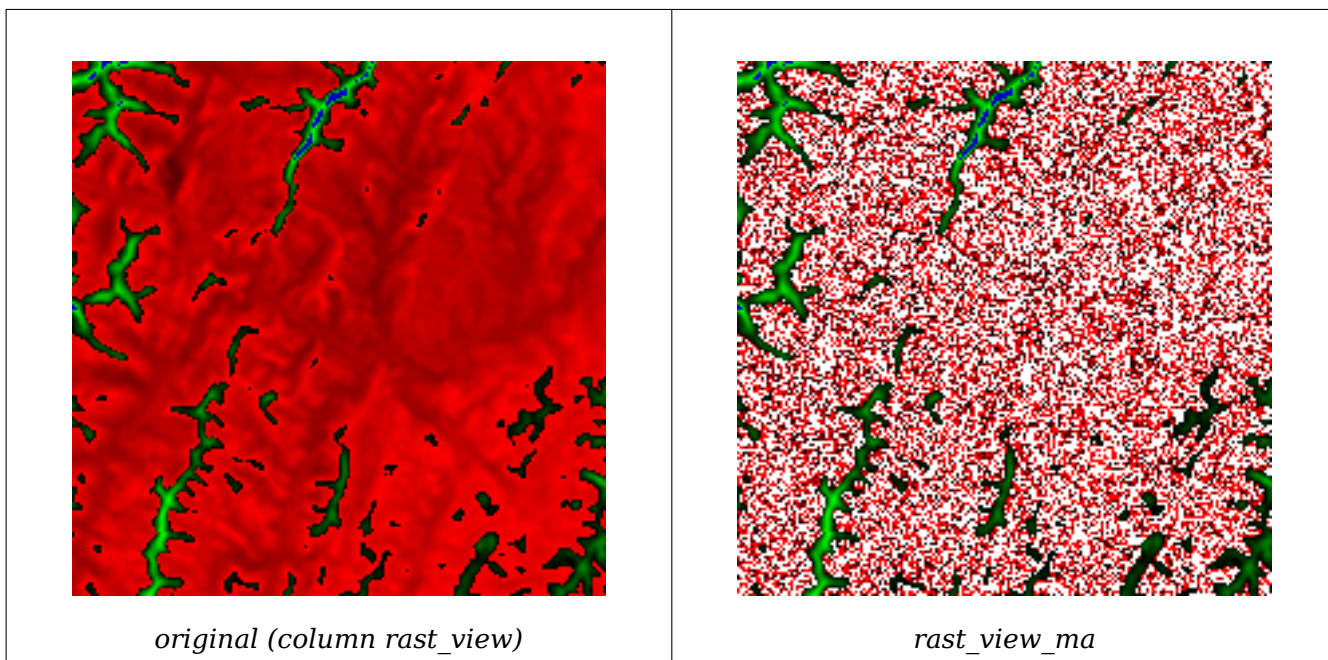


图 3 显示了使用 `ST_MapAlgebraExpr` 对栅格进行数学运算的结果。左侧是原始栅格，右侧是应用了 `tan([rast])*[rast]` 表达式后的结果，显示了明显的非线性变换效果。

```
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;
```

图 3

`ST_MapAlgebraExpr`, `ST_MapAlgebraFct`, `ST_BandPixelType`, `ST_GeoReference`, `ST_Value`

11.12.8 ST_MapAlgebraExpr

`ST_MapAlgebraExpr` — 对栅格进行数学运算。该函数是 PostgreSQL 的扩展，用于对栅格数据进行逐像素的数学运算。它接受两个栅格输入，一个表达式，以及可选的像素类型和扩展类型。表达式可以是任意的 SQL 表达式，用于对输入栅格的像素值进行计算。扩展类型可以是 `INTERSECTION`, `UNION`, `FIRST`, `SECOND` 等。

Synopsis

raster **ST_MapAlgebraExpr**(raster rast1, raster rast2, text expression, text pixeltype=same_as_rast1_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double 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);

警告



Warning

ST_MapAlgebraExpr 2.1.0 版本已弃用。请使用 **ST_MapAlgebra (expression version)** 版本。

rast1, (**rast2**) 表达式 2 个 raster，PostgreSQL 2.1.0 版本。band1, band2 指定 raster 的 band，band1 指定 rast1 的 band，band2 指定 rast2 的 band。extenttype 指定输出 raster 的 extenttype。

expression 2 个 PostgreSQL 表达式。PostgreSQL 表达式：((([rast1] + [rast2])/2.0)::integer)

pixeltype 指定输出 raster 的 pixeltype。ST_BandPixelType 指定输出 raster 的 pixeltype，NULL 指定输出 raster 的 pixeltype 与输入 raster 的 pixeltype 相同。

extenttype 指定输出 raster 的 extenttype。

1. INTERSECTION - 指定输出 raster 的 extenttype 为交集。
2. UNION - 指定输出 raster 的 extenttype 为并集。
3. FIRST - 指定输出 raster 的 extenttype 为第一个 raster 的 extenttype。
4. SECOND - 指定输出 raster 的 extenttype 为第二个 raster 的 extenttype。

nodata1expr rast1 的 NODATA 值。rast2 的 NODATA 值。rast2 的 NODATA 值。

nodata2expr rast2 的 NODATA 值。rast1 的 NODATA 值。rast1 的 NODATA 值。

nodatanodataval rast1 与 rast2 的 NODATA 值。

pixeltype 指定输出 raster 的 pixeltype。pixeltype 为 NULL 指定输出 raster 的 pixeltype 与输入 raster 的 pixeltype 相同。

[rast1.val], [rast2.val], [rast1.x], [rast1.y] 指定输出 raster 的 pixeltype。

2.0.0 版本。

警告：2 个 raster

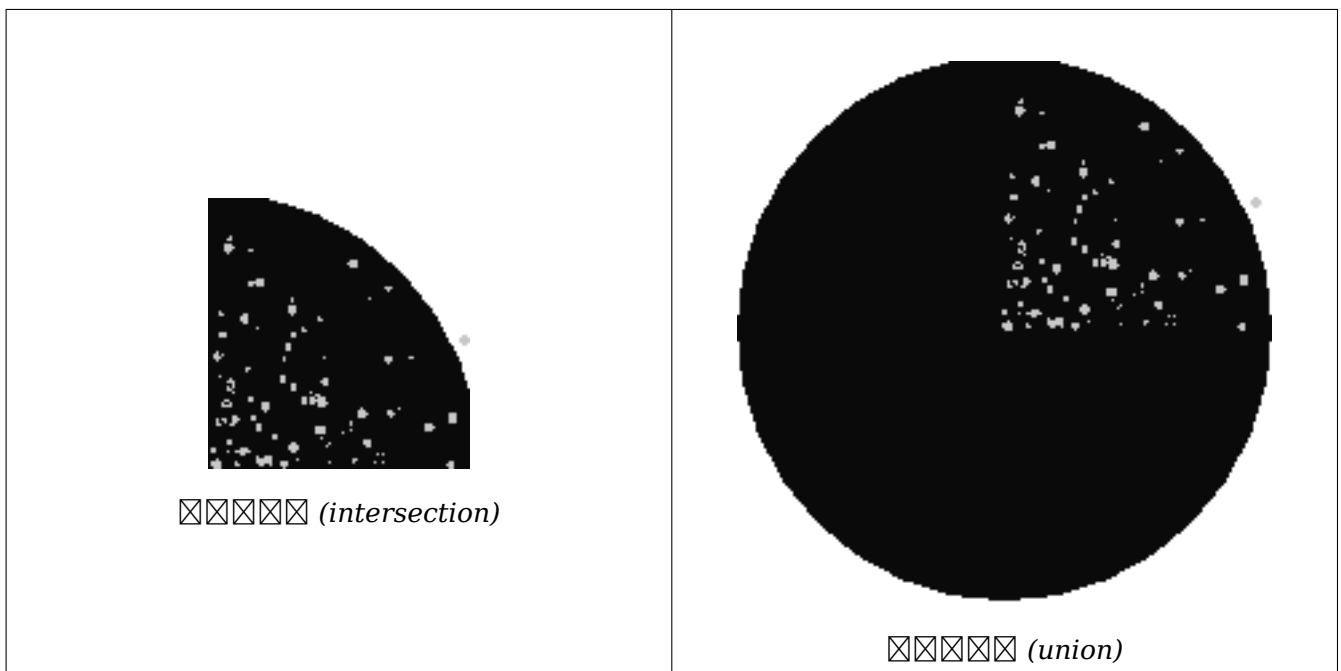
2 个 raster (modulo) 1 个 raster。

```
--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
```



图例: 图例图例图例图例图例图例图例图例图例图例

```
-- 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;
```

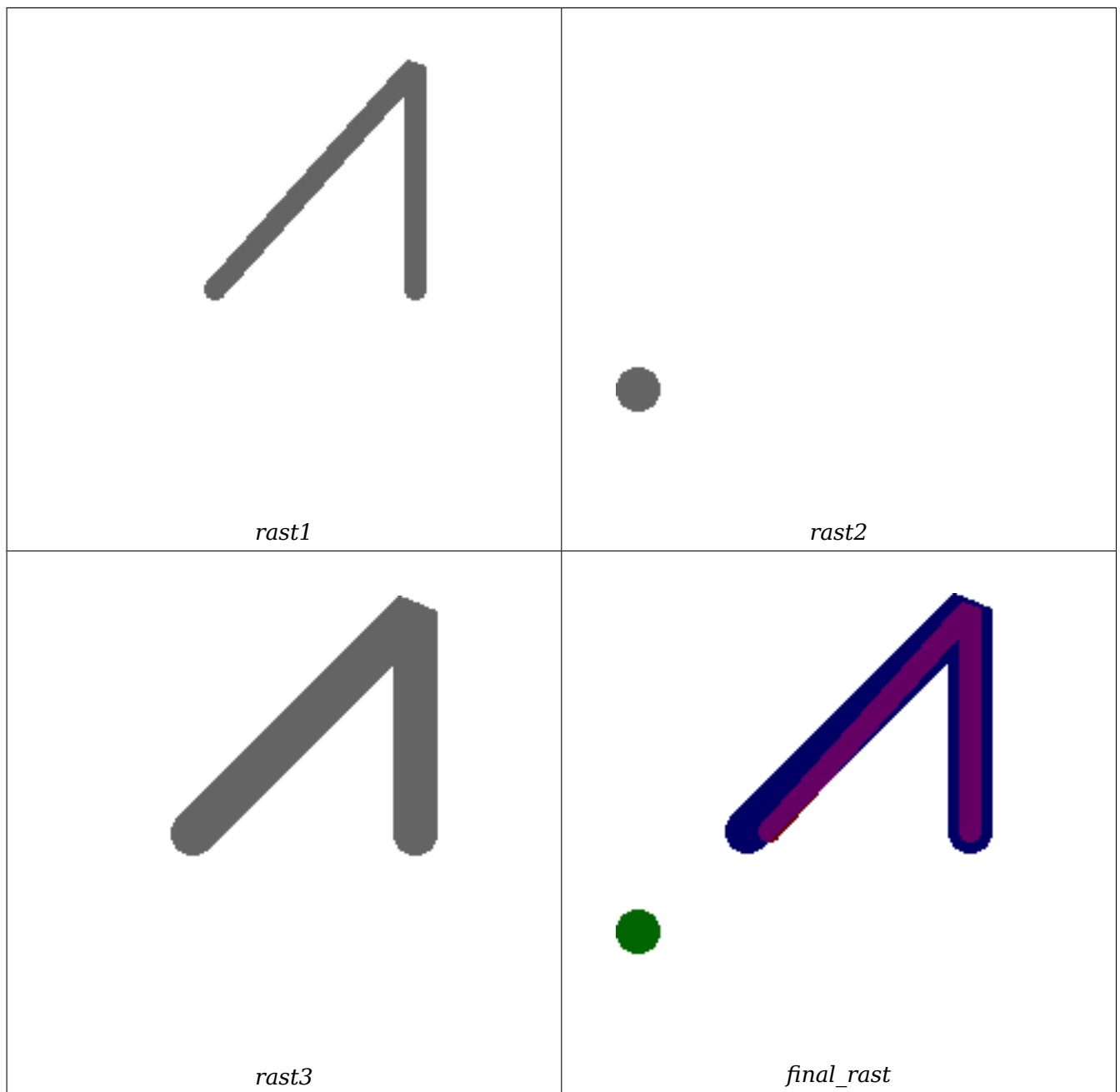


图 10-10: 使用 ST_Union 和 ST_Intersection 创建最终栅格

```
-- 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);
```

警告



Warning

ST_MapAlgebraFct 2.1.0 版本中，**ST_MapAlgebra** (callback function version) 已被弃用。

rast (rast) 是 **onerasteruserfunc** 在 PostgreSQL 中注册的函数名，**band** 是 1 到 **rast** 的带号。如果 **band** 为 1，则使用 **rast** 的所有带。如果 **band** 为 1，则使用 **rast** 的所有带。

pixeltype 是像素的数据类型。如果 **pixeltype** 为 NULL，则使用 **rast** 的像素数据类型。

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 regprodedure 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 是用户函数的名称。如果 **ST_MapAlgebraFct** 使用 **userfunction**，args 是变长参数。

**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` 的 `userfunction` 参数 `text[]` 是必需的。

2.0.0 版本中的示例。

SQL

创建 2 个栅格 (modulo) 函数，将 1 个栅格转换为 2 个栅格。

```
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
254	0
250	0
254	0
254	0

创建 NODATA 函数 (0) 将 2BUI 栅格转换为 1 个栅格。

```
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
```

```

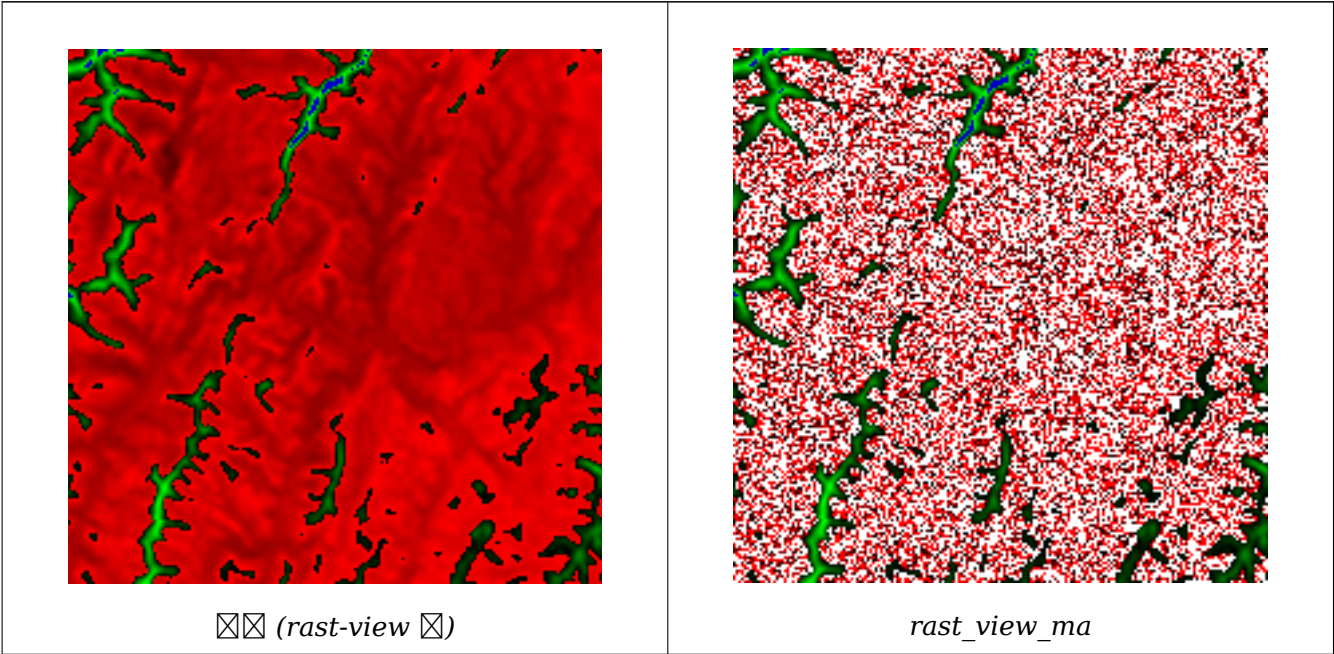
        noata := args[1];
    END IF;
    IF pixel < 251 THEN
        RETURN 1;
    ELSIF pixel = 252 THEN
        RETURN 2;
    ELSIF pixel
> 252 THEN
        RETURN 3;
    ELSE
        RETURN noata;
    END IF;
END;
$$
LANGUAGE 'plpgsql';
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ↵
[],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
```



`rast1`, `rast2` 是 `tworastuserfunc` 在 PostgreSQL 中定义的函数，第 1 个参数是 `rast1`，第 2 个参数是 `rast2`，第 3 个参数是 `band1`，第 4 个参数是 `band2`，第 5 个参数是 `pos`。函数返回一个栅格值。

`pixeltype` 是栅格的像素数据类型。如果 `pixeltype` 是 NULL，则返回 `rast1` 的像素数据类型。

The `tworastuserfunc` parameter must be the name and signature of an SQL or PL/pgSQL function, cast to a regprocedure. An example PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION simple_function_for_two_rasters(pixel1 FLOAT, pixel2 FLOAT, pos
    INTEGER[], VARIADIC args TEXT[])
    RETURNS FLOAT
    AS $$ BEGIN
        RETURN 0.0;
    END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

The `tworastuserfunc` may accept three or four arguments: a double precision value, a double precision value, an optional integer array, and a variadic text array. The first argument is the value of an individual raster cell in `rast1` (regardless of the raster datatype). The second argument is an individual raster cell value in `rast2`. The third argument is the position of the current processing cell in the form `'{x,y}'`. The fourth argument indicates that all remaining parameters to `ST_MapAlgebraFct` shall be passed through to the `tworastuserfunc`.

Passing a regprocedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'simple_function(double precision, double precision, integer[], text[])::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

The fourth argument to the `tworastuserfunc` is a variadic text array. All trailing text arguments to any `ST_MapAlgebraFct` call are passed through to the specified `tworastuserfunc`, and are contained in the `userargs` argument.



Note

(`VARIADIC`) 是 PostgreSQL 中的一个关键字，用于定义变长参数。有关更多信息，请参阅 [Query Language \(SQL\) Functions](#) 中的“SQL Functions with Variable Numbers of Arguments”部分。



Note

在 `tworastuserfunc` 中，`text[]` 是一个变长文本数组。

2.0.0 版本中已弃用。

示例：

```
-- define our user defined function --
CREATE OR REPLACE FUNCTION raster_mapalgebra_union(
    rast1 double precision,
```

```

    rast2 double precision,
    pos integer[],
    VARIADIC userargs text[]
)
RETURNS double precision
AS $$
DECLARE
BEGIN
    CASE
        WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
            RETURN ((rast1 + rast2)/2.);
        WHEN rast1 IS NULL AND rast2 IS NULL THEN
            RETURN NULL;
        WHEN rast1 IS NULL THEN
            RETURN rast2;
        ELSE
            RETURN rast1;
    END CASE;

    RETURN NULL;
END;
$$ LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- prep our test table of rasters
DROP TABLE IF EXISTS map_shapes;
CREATE TABLE map_shapes(rid serial PRIMARY KEY, rast raster, bnum integer, descrip text);
INSERT INTO map_shapes(rast,bnum, descrip)
WITH mygeoms
    AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(90,90),30) As geom, 'circle' As descrip
        UNION ALL
        SELECT 3 AS bnum,
            ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 15) As geom, ←
            'big road' As descrip
        UNION ALL
        SELECT 1 As bnum,
            ST_Translate(ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'), ←
            8,'join=bevel'), 10,-6) As geom, 'small road' As descrip
    ),
-- define our canvas to be 1 to 1 pixel to geometry
canvas
    AS ( SELECT ST_AddBand(ST_MakeEmptyRaster(250,
        250,
        ST_XMin(e)::integer, ST_YMax(e)::integer, 1, -1, 0, 0 ) , '8BUI'::text,0) As rast
        FROM (SELECT ST_Extent(geom) As e,
            Max(ST_SRID(geom)) As srid
            from mygeoms
        ) As foo
    )
-- return our rasters aligned with our canvas
SELECT ST_AsRaster(m.geom, canvas.rast, '8BUI', 240) As rast, bnum, descrip
FROM mygeoms AS m CROSS JOIN canvas
UNION ALL
SELECT canvas.rast, 4, 'canvas'
FROM canvas;

-- Map algebra on single band rasters and then collect with ST_AddBand
INSERT INTO map_shapes(rast,bnum,descrip)
SELECT ST_AddBand(ST_AddBand(rasts[1], rasts[2]),rasts[3]), 4, 'map bands overlay fct union ←
    (canvas)'
FROM (SELECT ARRAY(SELECT ST_MapAlgebraFct(m1.rast, m2.rast,
    'raster_mapalgebra_union(double precision, double precision, integer[], text[]) ←
    '::regprocedure, '8BUI', 'FIRST')

```

```
FROM map_shapes As m1 CROSS JOIN map_shapes As m2
WHERE m1.descrip = 'canvas' AND m2.descrip <
> 'canvas' ORDER BY m2.bnum) As rasts) As foo;
```



图例 (R: small road, G: circle, B: big road)

图例 (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 \(Map Algebra Nearest Neighbor\) \(neighborhood\) PostgreSQL 12.1](#).

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) .

neighborhood 1 参数: 邻域 (neighborhood) 参数 PostgreSQL 函数 `ST_Average` 的别名。邻域参数指定了用于计算平均值的像素范围。默认值为 1，表示 3x3 邻域。邻域参数可以是整数或字符串。如果为字符串，则必须与 `ST_Average` 函数兼容。

rast 栅格 (raster) 数据类型。

band 带 (band) 参数 (默认值 1)

pixeltype 像素类型 (pixel type)。参数 `ST_BandPixelType` 返回指定带的像素类型。如果像素类型为 `NULL`，则返回 `NULL`。如果像素类型为 `rast`，则返回 `rast` 的像素类型。如果像素类型为 `band`，则返回 `band` 的像素类型。

ngbwidth 邻域宽度 (neighborhood width) 参数

ngbheight 邻域高度 (neighborhood height) 参数

onerastngbuserfunc 邻域用户函数 (one raster neighborhood user function) 参数 PL/pgSQL 函数 `psql` 的别名。邻域用户函数参数指定了用于计算平均值的用户函数。默认值为 2，表示 3x3 邻域。

nodatamode NODATA 或 NULL 模式 (nodatamode) 参数

'ignore': 忽略 NODATA 值。如果输入栅格包含 NODATA 值，则输出栅格中的相应位置也将为 NODATA。

'NULL': 将 NODATA 值转换为 NULL。如果输入栅格包含 NODATA 值，则输出栅格中的相应位置将为 NULL。

'value': 将 NODATA 值转换为指定的值 (默认值为 0)。如果输入栅格包含 NODATA 值，则输出栅格中的相应位置将为指定的值。

args 参数列表 (args) 参数

2.0.0 版本 (2.0.0 version) 参数

函数

参考 http://trac.osgeo.org/gdal/wiki/frmts_wtkraster.html 中的 `ST_Rescale` 函数。

```
--
-- A simple 'callback' user function that averages up all the values in a neighborhood.
--
CREATE OR REPLACE FUNCTION rast_avg(matrix float[][], nodatamode text, variadic args text ↔
[])
RETURNS float AS
$$
DECLARE
    _matrix float[][];
    x1 integer;
    x2 integer;
    y1 integer;
    y2 integer;
    sum float;
BEGIN
    _matrix := matrix;
    sum := 0;
    FOR x in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
        FOR y in array_lower(matrix, 2)..array_upper(matrix, 2) LOOP
            sum := sum + _matrix[x][y];
        END LOOP;
    END LOOP;
    RETURN (sum*1.0/(array_upper(matrix,1)*array_upper(matrix,2) )::integer ;
```

```

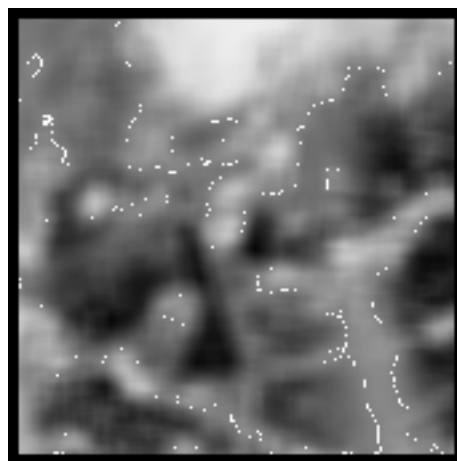
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- now we apply to our raster averaging pixels within 2 pixels of each other in X and Y ↔
direction --
SELECT ST_MapAlgebraFctNgb(rast, 1, '8BUI', 4,4,
    'rast_avg(float[][], text, text[])::regprocedure, 'NULL', NULL) As nn_with_border
FROM katrinas_rescaled
limit 1;

```



katrinas_rescaled



katrinas_rescaled 4x4 邻域平均结果

图 11.12.12

[ST_MapAlgebraFct](#), [ST_MapAlgebraExpr](#), [ST_Rescale](#)

11.12.12 ST_Reclass

ST_Reclass — 对栅格数据进行重新分类。nband 参数指定栅格的波段数。band 参数指定要重新分类的波段。reclassarg 参数指定重新分类的表达式。pixeltype 参数指定重新分类后的数据类型。示例：将 16BUI 栅格重新分类为 8BUI 栅格。

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);

```

图 11.12.13

reclassarg (rast) reclassexpr PostgreSQL 表达式。band 参数指定要重新分类的波段。band 参数指定要重新分类的波段。band 参数指定要重新分类的波段。band 参数指定要重新分类的波段。

。 使用 `reclassarg` 。

使用 `pixeltype` 。

2.0.0 。

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

2.0.0 版本中引入。

注意: 2.1.0 版本中引入 (即 C 语言版本)。

2.1.0 版本中引入 `ST_Union(rast, unionarg)` 函数。

注意: 2.1.0 版本中 `ST_Union(rast)` 仅 1 个参数。PostGIS 2.0 版本中引入。

注意: 2.1.0 版本中 `ST_Union(rast, uniontype)` 仅 4 个参数。

例: 创建单波段栅格

```
-- 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;
```

例: 创建多波段栅格

```
-- 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) ←
);
```

例: 创建多波段栅格

注意: 2.1.0 版本中引入, 2.0.0 版本中引入。

```
-- this creates a multi band raster collecting all the tiles that intersect a line
SELECT ST_Union(rast,ARRAY[ROW(2, 'LAST'), ROW(1, 'LAST'), ROW(3, 'LAST')]::unionarg[])
FROM aerials.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

例

`unionarg`, `ST_Envelope`, `ST_ConvexHull`, `ST_Clip`, `ST_Union`

11.13 空间函数

11.13.1 ST_Distinct4ma

`ST_Distinct4ma` — 计算 4 个移动平均值的距离。

Synopsis

float8 **ST_Distinct4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Distinct4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);



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_distinct4ma(float[],text,text[])':: ↵
    regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |          3
(1 row)
```

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 —

Synopsis

double precision **ST_InvDistWeight4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

几何

ST_InvDistWeight4ma (Inverse Distance Weighted method) 使用反距离加权法。

userargs 是包含 2 个元素的数组。第一个元素是力率 (power factor) k (范围从 0 到 1)。第二个元素是 1 到 4 的整数，表示要使用的最近邻的数量。默认值是 4。

返回类型是 double precision。

$$\hat{z}(x_o) = \frac{\sum_{j=1}^m z(x_j) d_{ij}^{-k}}{\sum_{j=1}^m d_{ij}^{-k}}$$

k = 力率 (power factor), 0 到 1 之间的实数



Note

ST_MapAlgebra (callback function version) 使用回调函数版本。

2.1.0 版本引入。

几何

-- NEEDS EXAMPLE

几何

ST_MapAlgebra (callback function version), **ST_MinDist4ma**

11.13.3 ST_Max4ma

ST_Max4ma — 使用最近 4 个邻居的最大值。

Synopsis

float8 **ST_Max4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Max4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

例。

ST_MapAlgebraFctNgb 函数。

例 2 中，ST_MapAlgebra 的 userargs 参数为 NODATA 函数。



Note

例 1 中 ST_MapAlgebraFctNgb 函数。



Note

例 2 中 ST_MapAlgebra (callback function version) 函数。



Warning

2.1.0 版本中 ST_MapAlgebraFctNgb 函数。

2.0.0 版本中。

例：2.1.0 版本中 2 个函数。

例。

```
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)
```

例。

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 — 计算 4 个移动平均值的平均值。

Synopsis

float8 **ST_Mean4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Mean4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

注意

ST_Mean4ma 返回一个 float8 值。

参数 2 是 pos，是 userargs 中 NODATA 模式的位置。



Note

参数 1 是 ST_MapAlgebraFctNgb 的回调函数。



Note

参数 2 是 ST_MapAlgebra (callback function version) 的回调函数。



Warning

2.1.0 版本中 ST_MapAlgebraFctNgb 的回调函数 1 是必需的。

2.0.0 版本中 ST_Mean4ma 返回一个 float8 值。

注意: 2.1.0 版本中 ST_Mean4ma 返回一个 double precision 值。

示例: 示例 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)
```

示例: 示例 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]

```

2  ,  userargs  NODATA

```



Note

1 ST_MapAlgebraFctNgb



Note

2 ST_MapAlgebra (callback function version)



Warning

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_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_Min4ma, ST_Maxima, ST_Minima, ST_Min4ma, ST_Max4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

11.13.6 ST_MinDist4ma

[illegible]

Synopsis

```
double precision ST_MinDist4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);
```

[illegible]

Note!

Note
 ST_InvDistWeight4ma
 .



Note
 ST_MapAlgebra (callback function version)
 .

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.

-- NEEDS EXAMPLE

返回

[ST_MapAlgebra \(callback function version\)](#), [ST_InvDistWeight4ma](#)

11.13.7 ST_Range4ma

ST_Range4ma — 返回栅格中指定范围的4邻域平均值。

Synopsis

float8 **ST_Range4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Range4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

返回

返回栅格中指定范围的4邻域平均值。

参数 2 个, 用户参数 userargs 个 NODATA 值。



Note

参数 1 个 [ST_MapAlgebraFctNgb](#) 函数。



Note

参数 2 个 [ST_MapAlgebra \(callback function version\)](#) 函数。



Warning

2.1.0 版本 [ST_MapAlgebraFctNgb](#) 函数参数 1 个。

2.0.0 版本。

版本: 2.1.0 版本 2 个。

返回

```
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)
```

返回

[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` — 返回指定栅格中每个像素的 4 邻域标准差。

Synopsis

`float8 ST_StdDev4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);`
`double precision ST_StdDev4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);`

返回

返回指定栅格中每个像素的 4 邻域标准差。



Note

参数 1 是 [ST_MapAlgebraFctNgb](#) 的 4 邻域标准差。



Note

参数 2 是 [ST_MapAlgebra](#) (callback function version) 的 4 邻域标准差。



Warning

2.1.0 版本中 [ST_MapAlgebraFctNgb](#) 的 4 邻域标准差。

2.0.0 版本中。

版本: 2.1.0 版本中 2 版本。

返回

```
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)
```

返回

[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` — 返回栅格数据中指定位置的4个相邻栅格单元值的总和。

Synopsis

`float8 ST_Sum4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);`
`double precision ST_Sum4ma(double precision[][] value, integer[][] pos, text[] VARIADIC userargs);`

返回

返回栅格数据中指定位置的4个相邻栅格单元值的总和。

参数 2 个, 用户参数 `userargs` 个 `NODATA` 模式。



Note

参数 1 个 `ST_MapAlgebraFctNgb` 返回栅格数据中指定位置的4个相邻栅格单元值的总和。



Note

参数 2 个 `ST_MapAlgebra` (callback function version) 返回栅格数据中指定位置的4个相邻栅格单元值的总和。



Warning

2.1.0 版本 `ST_MapAlgebraFctNgb` 返回栅格数据中指定位置的4个相邻栅格单元值的总和。

2.0.0 版本返回栅格数据中指定位置的4个相邻栅格单元值的总和。

版本: 2.1.0 版本 2 个参数。

返回

```
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)
```

函数

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra](#) (callback function version), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Mean4ma](#), [ST_Range4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

11.14 地形函数

11.14.1 ST_Aspect

`ST_Aspect` — 计算给定栅格的方位角 (以度为单位)。方位角是相对于正北方向测量的。

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);

函数

`ST_Aspect` 返回一个栅格，其值表示方位角。方位角是相对于正北方向测量的。

`units` 可以是 `RADIANS` 或 `DEGREES`。默认值是 `DEGREES`。

`units = RADIANS` 时，方位角的范围是 0 到 2π 。

`units = DEGREES` 时，方位角的范围是 0 到 360。

方位角为 0 表示正北，方位角为 -1 表示 NaN。



Note

方位角 (slope), 方位角 (aspect), 阴影 (hillshade) 函数在 [ESRI - How hillshade works](#) 和 [ERDAS Field Guide - Aspect Images](#) 中有详细解释。

2.0.0 版本中，方位角函数被重新实现。

更新: 2.1.0 版本中，`ST_MapAlgebra()` 函数被弃用，`interpolate_nodata` 参数被移除。

更新: 2.1.0 版本中，方位角函数的实现与 2.1.0 版本中的实现一致。

示例: 见 **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_Aspect(rast, 1, '32BF'))
FROM foo

```

```

-----
(1,"{{315,341.565063476562,0,18.4349479675293,45},{288.434936523438,315,0,45,71.5650482177734},{270
2227,180,161.565048217773,135}}")
(1 row)

```

11.14.1 ST_DumpValues

ST_DumpValues 返回栅格值。PostgreSQL 9.1 引入。

```

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;

```

11.14.2 ST_HillShade

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_HillShade, ST_Slope

11.14.2 ST_HillShade

ST_HillShade — 计算地形阴影。PostgreSQL 9.1 引入。

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);

参数

rast: 输入栅格, 可以是任意像素类型。如果像素类型不是 32BF, 则栅格值将被转换为 32BF。如果 **interpolate_nodata** 为 TRUE, 则栅格中的 NODATA 值将被插值。

band: 输入栅格的波段编号, 默认为 1。

azimuth: 太阳方位角, 默认为 315 (天顶)。

altitude: 太阳高度角, 默认为 45。

max_bright: 最大亮度, 默认为 255。

scale: 亮度比例, 默认为 1.0。如果 **scale** 为 0, 则亮度将被归一化到 0 到 255 的范围。

interpolate_nodata: 是否对 NODATA 值进行插值。如果为 TRUE, 则使用 **ST_InvDistWeight4ma** 函数对 NODATA 值进行插值。



Note

有关 Hillshade 的工作原理, 请参见 [How hillshade works](#)。

2.0.0 版本中的变化。

2.1.0 版本中的变化: **ST_MapAlgebra()** 函数, **interpolate_nodata** 参数。

2.1.0 版本中的变化: **ST_Hillshade()** 函数, **interpolate_nodata** 参数。

示例: 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[5][5]
  ) 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)

```

图 11.14.2

图 11.14.3 展示了使用 PostgreSQL 9.1 的 ST_Hillshade 函数。

```

WITH foo AS (
  SELECT ST_Tile(
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
        1, '32BF', 0, -9999
      ),
      1, 1, 1, ARRAY[
        [1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 2, 1],
        [1, 2, 2, 3, 3, 1],
        [1, 1, 3, 2, 1, 1],
        [1, 2, 2, 1, 2, 1],
        [1, 1, 1, 1, 1, 1]
      ]::double precision[]
    ),
    2, 2
  ) AS rast
)
SELECT
  t1.rast,
  ST_Hillshade(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;

```

图 11.14.3

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_Aspect, ST_Slope

11.14.3 ST_Roughness

ST_Roughness — DEM 粗糙度 (roughness) 的栅格。

Synopsis

raster **ST_Roughness**(raster rast, integer nband, raster customextent, text pixeltype="32BF", boolean interpolate_nodata=FALSE);

¶

DEM 的 值 范围 在 0 到 255 之间。
2.1.0 版本 支持 64 位 整数。

¶

-- needs examples

¶

[ST_MapAlgebra \(callback function version\)](#), [ST_TRI](#), [ST_TPI](#), [ST_Slope](#), [ST_HillShade](#), [ST_Aspect](#)

11.14.4 ST_Slope

ST_Slope — 计算 坡度 (slope) 值。返回 坡度 值。

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);

¶

units 指定 坡度 值的 单位。可以是 RADIANS, DEGREES(度), PERCENT 百分比。

scale 指定 坡度 值的 比例尺。例如: scale=370400, 表示: 1 度 等于 370400 米。

scale=111120 表示: 1 度 等于 111120 米。

interpolate_nodata 指定 是否 对 坡度 值 进行 插值。如果 为 TRUE, 则 使用 [ST_InvDistWeight4ma](#) 函数 对 坡度 值 进行 插值。如果 为 FALSE, 则 使用 NODATA 值。



Note

坡度 (slope), 坡度 (aspect), 坡度 (hillshade) 的计算 方法 可以参考 [ESRI - How hillshade works](#) 和 [ERDAS Field Guide - Slope Images](#)。

2.0.0 版本 支持 64 位 整数。

版本: 2.1.0 版本 支持 ST_MapAlgebra() 函数, 支持 units, scale, interpolate_nodata 参数。

版本: 2.1.0 版本 支持 64 位 整数。2.1.0 版本 支持 64 位 整数。

实验: 实验 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)

```

实验: 实验 2

实验实验实验实验实验实验实验实验. 实验实验 PostgreSQL 9.1 实验实验实验实验实验实验实验实验.

```

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;
```

[ST_MapAlgebra \(callback function version\)](#), [ST_TRI](#), [ST_TPI](#), [ST_Roughness](#), [ST_HillShade](#), [ST_Aspect](#)

11.14.5 ST_TPI

`ST_TPI` — (Topographic Position Index)

Synopsis

raster **ST_TPI**(raster rast, integer nband, raster customextent, text pixelttype="32BF" , boolean interpolate_nodata=FALSE);

Calculates the Topographic Position Index, which is defined as the focal mean with radius of one minus the center cell.



Note
1 (focalmean radius of one)

2.1.0

-- needs examples

[ST_MapAlgebra \(callback function version\)](#), [ST_TRI](#), [ST_Roughness](#), [ST_Slope](#), [ST_HillShade](#), [ST_Aspect](#)

11.14.6 ST_TRI

`ST_TRI` — (Terrain Ruggedness Index)

Synopsis

raster **ST_TRI**(raster rast, integer nband, raster customextent, text pixelttype="32BF" , boolean interpolate_nodata=FALSE);

地形

地形粗糙度指数 (Terrain Ruggedness Index) 地形粗糙度指数。



Note

地形粗糙度指数 1 地形粗糙度指数 (focalmean radius of one) 地形粗糙度指数。

2.1.0 地形粗糙度指数。

地形

-- needs examples

地形

ST_MapAlgebra (callback function version), ST_Roughness, ST_TPI, ST_Slope, ST_HillShade, ST_Aspect

11.15 地形分析

11.15.1 Box3D

Box3D — 地形分析 BOX3D 地形分析。

Synopsis

box3d **Box3D**(raster rast);

地形

地形分析地形分析地形分析地形分析。

地形分析地形分析地形分析 ((MINX, MINY), (MAXX, MAXY)) 地形分析地形分析。

地形分析: 2.0.0 地形分析地形分析 BOX3D 地形分析 BOX2D 地形分析地形分析。 BOX2D 地形分析地形分析地形分析地形分析, 2.0.0 地形分析 BOX3D 地形分析地形分析。

地形

```
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)

¶¶

ST_Envelope

11.15.2 ST_ConvexHull

ST_ConvexHull — BandNoDataValue 返回凸包, 返回凸包。 返回凸包, ST_Envelope 返回凸包。 返回凸包。

Synopsis

geometry **ST_ConvexHull**(raster rast);

¶¶

NoDataBandValue 返回凸包, 返回凸包。 返回凸包, ST_Envelope 返回凸包。



Note

ST_Envelope 返回凸包 (floor) 返回凸包。 返回凸包 ST_ConvexHull 返回凸包。

¶¶

PostGIS Raster Specification 返回凸包。

```
-- 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) )
```



ST_Envelope, ST_MinConvexHull, ST_ConvexHull, ST_AsText

11.15.3 ST_DumpAsPolygons

ST_DumpAsPolygons — `geomval(geom, val)` `geomval(geom, val)`. `geomval(geom, val)`
`geomval(geom, val)` 1 `geomval(geom, val)`.

Synopsis

```
setof geomval ST_DumpAsPolygons(raster rast, integer band_num=1, boolean exclude_nodata_value=TRUE)
```



geomval (SRF; Set-Returning Function) geom (geom) val (val) geomval
geomval geomval. geomval val geomval
geomval.

ST_DumpAsPolygon 返回多边形集合。 使用 GROUP BY 子句按属性分组。

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

GDAL 1.7 ☒☒☒☒☒☒☒☒☒.



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!

Note

Note

XXXXXXXXXXXXXXXXXXXXXXXXXXXXX, ST ValueCount XXXXXXXXXX.



Note

Note

ST PixelAsPolygons .



```
-- 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
```


Synopsis

geometry **ST_MinConvexHull**(raster rast, integer nband=NULL);

返回

返回 NODATA 如果输入栅格包含 NODATA 值。 nband 如果 NULL, 返回与输入栅格相同的栅格。

2.1.0 版本引入。

示例

```
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 0,9 0,9 -9,0 -9,0 0))	POLYGON((0 -3,9 -3,9 -9,0 -9,0 -3))	POLYGON((3 -3,9 -3,9 -6,3 -6,3 -3))	POLYGON((0 -3,6 -3,6 -9,0 -9,0 -3))


```
-- Or if you want the no data value different for just one time

SELECT ST_AsText(
    ST_Polygon(
        ST_SetBandNoDataValue(rast,1,252)
    ) As geomwkt
FROM dummy_rast
WHERE rid =2;

geomwkt
-----
MULTIPOLYGON(((3427928 5793243.85,3427928 5793243.8,3427928 5793243.75,3427927.85 ↵
5793243.75,3427927.8 5793243.75,3427927.8 5793243.8,3427927.75 5793243.8,3427927.75 ↵
5793243.85,3427927.75 5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.85 ↵
5793244,3427927.9 5793244,3427928 5793244,3427928 5793243.95,3427928 5793243.85) ↵
,(3427927.9 5793243.9,3427927.9 5793243.85,3427927.95 5793243.85,3427927.95 ↵
5793243.9,3427927.9 5793243.9)))
```

☒☒

ST_Value, ST_DumpAsPolygons

11.16 简体中文版

11.16.1 &&

&& — A 与 B 的逐像素逻辑与。TRUE 当且仅当 A 和 B 都为 TRUE 时。

Synopsis

```
boolean &&( raster A , raster B );
boolean &&( raster A , geometry B );
boolean &&( geometry B , raster A );
```

☒☒

&& 对 A 和 B 的每个像素执行逻辑与操作。TRUE 当且仅当 A 和 B 都为 TRUE 时。



Note

☒☒☒☒ (operand) 必须是 2D 栅格或几何体。

2.0.0 版本引入。

例

```
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 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，如果 A 与 B 的交集不为 TRUE，则返回 FALSE。



Note

如果 A 或 B 为 NULL，则返回 NULL。

例

```
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 与 B 的并集是否为 TRUE。

Synopsis

boolean **&>**(raster A , raster B);

描述

&> 对栅格 A 和栅格 B 进行逐像素比较，如果 A 的像素值大于 B 的像素值，则返回 TRUE，否则返回 FALSE。



Note

该函数 (operand) 仅支持栅格数据类型。

示例

```
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 栅格与 B 栅格进行逐像素比较，如果 A 的像素值等于 B 的像素值，则返回 TRUE，否则返回 FALSE。

Synopsis

boolean **=**(raster A , raster B);

描述

= 对栅格 A 和栅格 B 进行逐像素比较，如果 A 的像素值等于 B 的像素值，则返回 TRUE。PostgreSQL 还支持 **=**, **<**, **>** 运算符 (如: GROUP BY 或 ORDER BY 子句)。



Caution

该函数 (operand) 仅支持栅格数据类型。该函数 **~** 运算符 (group by) 不支持。

2.1.0 版本引入。

返回

~ =

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 B 与 A 的交集是否为 TRUE。



Note

返回 (operand) 返回 TRUE 或 FALSE。

2.0.0 返回 raster @ raster, raster @ geometry 返回 TRUE 或 FALSE。

2.0.5 返回 geometry @ raster 返回 TRUE 或 FALSE。

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 A 与 B 的交集是否为 TRUE。



Note

返回 (operand) 返回 TRUE 或 FALSE。

2.0.0 返回 TRUE 或 FALSE。

例

将精度为 2 的栅格数据与精度为 1 的栅格数据相加。

```
SELECT ST_AddBand(prec.rast, alt.rast) As new_rast
FROM prec INNER JOIN alt ON (prec.rast ~= alt.rast);
```

例

ST_AddBand, =

11.16.7 ~

~ — A 与 B 的异或结果。如果 A 和 B 的像素值不同，则返回 TRUE。否则返回 FALSE。

Synopsis

```
boolean ~( raster A , raster B );
boolean ~( geometry A , raster B );
boolean ~( raster B , geometry A );
```

例

~ 返回 A 和 B 的异或结果。如果 A 和 B 的像素值不同，则返回 TRUE。否则返回 FALSE。



Note

~ 的左操作数 (operand) 必须是栅格数据。

2.0.0 版本引入。

例

@

11.17 空间关系函数

11.17.1 ST_Contains

ST_Contains — 如果栅格 rastA 包含栅格 rastB，则返回 TRUE。否则返回 FALSE。

Synopsis

```
boolean ST_Contains( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Contains( raster rastA , raster rastB );
```


[illegible]

Note

[illegible]

Note

ST_Contains(ST_Polygon(raster), geometry) ST_Contains(geometry, ST_Polygon(raster))



Note!

Note

`ST_Contains()` ☒ `ST_Within()` ☒☒☒☒☒☒☒. ☒☒☒, `ST_Contains(rastA, rastB)` ☒☒☒☒
`ST_Within(rastB, rastA)` ☒☒☒☒☒☒☒☒☒☒.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.



```
-- specified band numbers
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↵
    dummy_rast r2 WHERE r1.rid = 1;
```

NOTICE: The first raster provided has no bands

```

rid | rid | st_contains
-----+-----
  1 |  1 |
  1 |  2 | f

```

```
-- no band numbers specified
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ↵
    dummy_rast r2 WHERE r1.rid = 1;
```

```

rid | rid | st_contains
-----+-----
  1 |  1 | t
  1 |  2 | f

```



ST Intersects, ST Within

11.17.2 ST Contains Properly

[illegible]

Synopsis

```
boolean ST_ContainsProperly( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_ContainsProperly( raster rastA , raster rastB );
```

注意

如果 rastB 包含 rastA 且 rastA 与 rastB 的交集面积等于 rastA 与 rastB 的交集面积，则返回 true。如果 rastA 或 rastB 为 NULL，则返回 NULL。如果 rastA 或 rastB 包含 NODATA 值，则返回 NULL。

如果 rastA 与 rastB 的交集面积等于 rastA 的面积，则返回 true。



Note

如果 rastA 与 rastB 的交集面积等于 rastA 的面积，则返回 true。



Note

如果 rastA 与 rastB 的交集面积等于 rastA 的面积，则返回 true。ST_ContainsProperly(ST_Polygon(raster), geometry) 与 ST_ContainsProperly(geometry, ST_Polygon(raster)) 返回 true 当且仅当 ST_Polygon 包含 geometry。

2.1.0 版本引入。

注意

```
SELECT r1.rid, r2.rid, ST_ContainsProperly(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_containsproperly
2	1	f
2	2	f

注意

ST_Intersects, ST_Contains

11.17.3 ST_Covers

ST_Covers — 如果 rastB 包含 rastA 且 rastA 与 rastB 的交集面积等于 rastA 的面积，则返回 true。

Synopsis

```
boolean ST_Covers( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Covers( raster rastA , raster rastB );
```

注意

`ST_Covers(rastB, rastA)` 返回 `TRUE` 当且仅当 `rastA` 完全被 `rastB` 覆盖。如果 `rastA` 或 `rastB` 包含任何 `NULL` 值，则返回 `NULL`。如果 `rastA` 或 `rastB` 包含任何 `NODATA` 值，则返回 `NULL`。



Note

此函数在 PostGIS 3.6.0alpha1 中引入。



Note

此函数在 PostGIS 3.6.0alpha1 中引入。它使用 `ST_Covers(ST_Polygon(raster), geometry)` 和 `ST_Covers(geometry, ST_Polygon(raster))` 来检查 `ST_Polygon` 是否被覆盖。

2.1.0 版本中引入。

注意

```
SELECT r1.rid, r2.rid, ST_Covers(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_covers
2	1	f
2	2	t

注意

[ST_Intersects](#), [ST_CoveredBy](#)

11.17.4 ST_CoveredBy

`ST_CoveredBy` — 返回 `TRUE` 当且仅当 `rastA` 完全被 `rastB` 覆盖。

Synopsis

```
boolean ST_CoveredBy( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_CoveredBy( raster rastA , raster rastB );
```

注意

`ST_CoveredBy(rastA, rastB)` 返回 `TRUE` 当且仅当 `rastA` 完全被 `rastB` 覆盖。如果 `rastA` 或 `rastB` 包含任何 `NULL` 值，则返回 `NULL`。如果 `rastA` 或 `rastB` 包含任何 `NODATA` 值，则返回 `NULL`。



Note

ST_CoveredBy(geometry, ST_Polygon(raster)) 与 ST_CoveredBy(ST_Polygon(raster), geometry) 等价。



Note

ST_CoveredBy(geometry, ST_Polygon(raster)) 与 ST_CoveredBy(ST_Polygon(raster), geometry) 等价。ST_CoveredBy(geometry, ST_Polygon(raster)) 与 ST_CoveredBy(ST_Polygon(raster), geometry) 等价。

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 rastA 或 rastB 为 NULL，则返回 NULL。如果 rastA 或 rastB 为 NODATA，则返回 false。



Note

ST_Disjoint(geometry, ST_Polygon(raster)) 与 ST_Disjoint(ST_Polygon(raster), geometry) 等价。

**Note**

ST_Disjoint(ST_Polygon(raster), geometry) 与 ST_Polygon 不同。

2.1.0 版本。

```

-- 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;

```

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

```

```


```

ST_Intersects**11.17.6 ST_Intersects**

ST_Intersects — 检查 rasterA 与 rasterB 是否相交。

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 );

```

```


```

检查 rasterA 与 rasterB 是否相交。如果任一输入为 NULL 或 0，则返回 NULL。如果任一输入为 0，则返回 (NODATA)。

**Note**

ST_Intersects(geometry, raster) 和 ST_Intersects(raster, geometry) 是等效的。

PostGIS: 2.0.0 版本中引入/支持。

**Warning**

PostGIS: 2.1.0 版本中 ST_Intersects(geometry, raster) 和 ST_Intersects(raster, geometry) 是等效的。

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 — 检查 rasterA 和 rasterB 是否重叠。如果重叠，返回 true，否则返回 false。

Synopsis

```
boolean ST_Overlaps( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Overlaps( raster rastA , raster rastB );
```

SQL

检查 rasterA 和 rasterB 是否重叠。如果重叠，返回 true，否则返回 false。如果 rasterA 和 rasterB 都是 NULL，则返回 NULL。如果 rasterA 或 rasterB 是 (NODATA 值) 则返回 false。

**Note**

ST_Overlaps(geometry, raster) 和 ST_Overlaps(raster, geometry) 是等效的。

**Note**

ST_Overlaps(ST_Polygon(raster), geometry) 返回 TRUE 当且仅当 ST_Polygon 与 geometry 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 — 返回 TRUE 当且仅当 rastA 与 rastB 在边界处接触，但不重叠。

Synopsis

boolean **ST_Touches**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Touches**(raster rastA , raster rastB);

SQL

ST_Touches(rastA, nbandA, rastB, nbandB) 返回 TRUE 当且仅当 rastA 与 rastB 在边界处接触，但不重叠。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 TRUE 当且仅当 ST_Polygon 与 geometry 在边界处接触。

2.1.0 版本引入。

例

```
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

例

ST_Intersects

11.17.9 ST_SameAlignment

ST_SameAlignment — 检查两个栅格是否具有相同的对齐方式 (即栅格单元是否对齐)。如果两个栅格具有相同的对齐方式，则返回 true，否则返回 false。

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 );
```

例

检查两个栅格 (即 1, 2): (栅格单元是否对齐, 即, 栅格, SRID 是否相同) 是否具有相同的对齐方式。如果两个栅格具有相同的对齐方式，则返回 true，否则返回 false。 4 个栅格单元是否对齐 (NOTICE) 返回 true。

检查两个栅格 (即 3): 是否具有相同的对齐方式。 **ST_SameAlignment** 在 PostgreSQL 中被定义为聚合函数 (aggregate)。它使用 SUM() 和 AVG() 函数来计算栅格单元是否对齐。

2.0.0 版本中引入。

更新: 2.1.0 版本中引入。

例: 例

```
SELECT ST_SameAlignment(
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0)
) as sm;
```

```
sm
----
t
```



```
SELECT ST_SameAlignment(A.rast,b.rast)
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;

NOTICE: The two rasters provided have different SRIDs
NOTICE: The two rasters provided have different SRIDs
st_samealignment
-----
t
f
f
f
```

¶¶

Section [10.1](#), [ST_NotSameAlignmentReason](#), [ST_MakeEmptyRaster](#)

11.17.10 ST_NotSameAlignmentReason

ST_NotSameAlignmentReason — 返回两个栅格的不同对齐原因，返回原因代码的文本描述。

Synopsis

text **ST_NotSameAlignmentReason**(raster rastA, raster rastB);

¶¶

返回原因代码的文本描述，返回原因代码的文本描述。



Note

返回原因代码的文本描述，返回原因代码的文本描述 (返回原因代码的文本描述) 返回原因代码的文本描述。

2.1.0 返回原因代码的文本描述。

¶¶

```
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)
```

图例

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);

图例

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 NODATA 值，则返回 NULL。



Note
如果 operand 是 NULL，则返回 NULL。



Note
ST_Within(ST_Polygon(raster), geometry) 与 ST_Within(geometry, ST_Polygon(raster)) 返回相同的值。



Note
ST_Within() 与 ST_Contains() 返回相反的结果。即，ST_Within(rastA, rastB) 返回 true 当且仅当 ST_Contains(rastB, rastA) 返回 true。

2.1.0 版本中，ST_Within 返回 true。

图例

```
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 — 是否 rasterA 完全在 rasterB 内部。

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 );
```

参数

rastA, rastB 是栅格。nbandA, nbandB 是栅格的波段数。distance_of_srid 是距离，单位是 SRID 指定的单位。如果 distance_of_srid 是 NULL，则返回 TRUE。如果 distance_of_srid 是 0，则返回 FALSE。如果 distance_of_srid 是正数，则返回 TRUE。如果 distance_of_srid 是负数，则返回 FALSE。

如果 distance_of_srid 是正数，则返回 TRUE。如果 distance_of_srid 是负数，则返回 FALSE。如果 distance_of_srid 是 0，则返回 FALSE。如果 distance_of_srid 是 NULL，则返回 TRUE。



Note

如果 distance_of_srid 是 0，则返回 FALSE。



Note

ST_DFullyWithin(ST_Polygon(raster), geometry) 返回 TRUE 当且仅当 ST_Polygon 完全在 geometry 内部。

2.1.0 版本引入。

示例

```
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

相关链接

[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 地址标准化工具

lexicon **PAGC standardizer** 工具 (fork) 自 fork (从 PostgreSQL **PAGC PostgreSQL 标准化工具** 工具)。

lexicon 工具使用 lexicon (lexicon; lex) 和 gazetteer (gazetteer; gaz) 工具来生成地址标准化工具。

```
CREATE EXTENSION address_standardizer;
-- 安装 address_standardizer 工具
PostgreSQL 标准化工具。 address_standardizer 工具,
address_standardizer_data_us 工具, 标准化工具,
工具, 标准化工具。 CREATE EXTENSION address_standardizer_data_us;
-- 安装 address_standardizer_data_us 工具。
```

PostGIS extensions/address_standardizer 工具, 标准化工具, 标准化工具。

标准化工具 Section 2.3 标准化工具。

12.1.1 地址标准化工具

标准化工具, 标准化工具, 标准化工具 (macro) 标准化工具, 标准化工具 (micro) 标准化工具。 标准化工具, 标准化工具。

标准化工具 标准化工具/标准化工具, 标准化工具。

标准化工具/标准化工具 (**zip code**) 工具 (Perl) 标准化工具。 标准化工具 parseaddress-api.c 标准化工具, 标准化工具。

标准化工具 (Perl) 标准化工具。 标准化工具 parseaddress-api.c 标准化工具, 标准化工具 "includes" 标准化工具。

。

，。

id 。

rule 记录。 **PAGC Address Standardizer Rule records** 。

， -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)。

BUILDT 。

DIRECT (22)。

MILE (20)。

ROAD (6)。

RR (8)。

TYPE (2)。

pretype 类型 (枚举值 4) 名称: 街道前缀类型 (STREET PREFIX TYPE) 类型。

street 类型 (枚举值 5) 名称: 街道 (STREET NAME) 类型。

suftype 类型 (枚举值 6) 名称: St, Ave, Cir 街道后缀类型 (STREET POST TYPE) 类型。 示例: 75 State Street 类型 *STREET*

sufdir 类型 (枚举值 7) 名称: 街道后缀-方向性 (STREET POST-DIRECTIONAL) 类型。 示例: 3715 TENTH AVENUE WEST 类型 *WEST*

ARC_C

(枚举值 = "2")。HOUSE 类型 MICRO 类型。 示例: HOUSE 类型 MICRO_C 类型。

CIVIC_C

(枚举值 = "3")。HOUSE 类型。

EXTRA_C

(枚举值 = "4")。EXTRA 类型 - 类型 - 类型。 示例: 类型

EXTRA_C output tokens (excerpted from <http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ-->).

BLDNG (枚举值 0): 类型。

BOXH (token number 14): The **BOX** in BOX 3B

BOXT (枚举值 15): BOX 3B 类型 **3B**

RR (枚举值 8): RR 7 类型 **RR**

UNITH (枚举值 16): APT 3B 类型 **APT**

UNITT (枚举值 17): APT 3B 类型 **3B**

UNKNWN (枚举值 9): 类型。

12.1.3.2 lex table

lex table — 类型 (lex) 类型, 枚举值 (1) 类型 (the section called “**类型**” 类型) 类型 (2) 类型。

类型

类型 (lexicon) 类型, 枚举值 (1) 类型 (the section called “**类型**” 类型) 类型 (2) 类型。 示例: 类型, 类型, 类型 **ONE** 类型 stdword 1 类型

类型。 示例: 类型。

id 类型

seq 类型: 类型?

word 类型: 类型

stdword 类型: 类型

token 类型: 类型。 示例: 类型。 **PAGC Tokens** 类型

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 [X][X][X][X][X][X][X]

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, Waford, 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)						

❏

12.1.4.3 standardize_address

`standardize_address` — ❏, ❏, ❏ stdaddr ❏.

Synopsis

stdaddr **standardize_address**(text lextab, text gaztab, text rultab, text address);
stdaddr **standardize_address**(text lextab, text gaztab, text rultab, text micro, text macro);

❏

lex table, gaz table, rules table ❏ stdaddr ❏.

❏ 1: ❏.

❏ 2: ❏. house_num street ❏ micro
❏, city, state postal_code country ❏ macro ❏.

2.2.0 ❏.

✅ This method needs address_standardizer extension.

❏

address_standardizer_data_us ❏

```
CREATE EXTENSION address_standardizer_data_us; -- only needs to be done once
```

❏ 1: ❏. ❏.

```
SELECT house_num, name, suftype, city, country, state, unit FROM standardize_address(' ←
    us_lex',
                                           'us_gaz', 'us_rules', 'One Devonshire Place, PH 301, Boston, MA ←
                                           02109');
```

house_num	name	suftype	city	country	state	unit
1	DEVONSHIRE	PLACE	BOSTON	USA	MASSACHUSETTS	# PENTHOUSE 301

TIGER ❏ (❏ postgis_tiger_geocoder ❏ ❏.)

```
SELECT * FROM standardize_address('tiger.pgc_lex',
    'tiger.pgc_gaz', 'tiger.pgc_rules', 'One Devonshire Place, PH 301, Boston, MA ←
    02109-1234');
```

❏ hstore ❏. ❏ CREATE EXTENSION hstore; ❏.

```
SELECT (each(hstore(p))).*
FROM standardize_address('tiger.pgc_lex', 'tiger.pgc_gaz',
    'tiger.pgc_rules', 'One Devonshire Place, PH 301, Boston, MA 02109') As p;
```


key	value
box	
city	BOSTON
name	DEVONSHIRE
qual	
unit	# PENTHOUSE 301
extra	
state	MA
predir	
sufdir	
country	USA
pretype	
suftype	PL
building	
postcode	02109
house_num	1
ruralroute	

(16 rows)

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** 是 OpenStreetMap 数据的一个接口。它使用 osm2pgsql 工具，将 OpenStreetMap 数据导入 PostgreSQL 8.4 或 PostgreSQL 9.0 的 PostGIS 1.5 数据库中。Nominatim 使用 Tiger 数据，并提供了 SQL 接口。Nominatim 是 Tiger 数据的一个接口。它使用 SQL 接口，将 Tiger 数据导入 PostgreSQL 8.4 或 PostgreSQL 9.0 的 PostGIS 1.5 数据库中。
- **GIS Graphy** 是 PostGIS 的一个接口。它使用 OSM(OpenStreetMap) 数据，并提供了 SQL 接口。GIS Graphy 是 OSM 数据的一个接口。它使用 SQL 接口，将 OSM 数据导入 PostgreSQL 8.4 或 PostgreSQL 9.0 的 PostGIS 1.5 数据库中。

12.2.1 Drop_Indexes_Generate_Script

Drop Indexes Generate Script — TIGER 数据的一个接口。它使用 tiger_data 数据，并提供了 SQL 接口。

Synopsis

text **Drop_Indexes_Generate_Script**(text param_schema=tiger_data);

返回

TIGER 数据的一个接口。它使用 tiger_data 数据，并提供了 SQL 接口。

返回 (bloat) 数据。 **Install Missing Indexes** 数据的一个接口。它使用 tiger_data 数据，并提供了 SQL 接口。

2.0.0 版本。

返回

```
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_2010 和 tiger_2011 表的索引。

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_2010 和 tiger_2011 表的索引。

2.1.0 版本新增。

❏❏

```

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 state_all 表并生成 tiger_data 表的索引。

安装

PostGIS (PostgreSQL 扩展) 依赖于 NAD83 坐标系, 因此, 安装时, 需要安装 `normalized_address (addy)` 扩展。安装时, 需要安装 `normalized_address (addy)` 扩展。TIGER 数据 (edge, face, addr), PostgreSQL 索引 (soundex, levenshtein), 以及 PostGIS 扩展 TIGER 数据。安装时, 需要安装 `normalized_address (addy)` 扩展。安装时, 需要安装 `normalized_address (addy)` 扩展。安装时, 需要安装 `normalized_address (addy)` 扩展。

安装: 2.0.0 版本 TIGER 2010 数据, 安装时, 需要安装 `normalized_address (addy)` 扩展。安装时, 需要安装 `normalized_address (addy)` 扩展。安装时, 需要安装 `normalized_address (addy)` 扩展。

安装

安装 (MA), 安装 (MN), 安装 (CA), 安装 (RI) 以及 TIGER 数据 PostgreSQL 9.1rc1/PostGIS 2.0 版本 3.0 GHz 内存 2GB 内存 7 个数据库。

安装时, 需要安装 `normalized_address (addy)` 扩展。

```
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
```

安装时, 需要安装 `normalized_address (addy)` 扩展。

```
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
```

安装时, 需要安装 `normalized_address (addy)` 扩展。

```
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
```

安装时, 需要安装 `normalized_address (addy)` 扩展。

```
CREATE TABLE addresses_to_geocode(addid serial PRIMARY KEY, address text,
                                   lon numeric, lat numeric, new_address text, rating integer);
```

```
INSERT INTO addresses_to_geocode(address)
VALUES ('529 Main Street, Boston MA, 02129'),
```

```

('77 Massachusetts Avenue, Cambridge, MA 02139'),
('25 Wizard of Oz, Walaford, KS 99912323'),
('26 Capen Street, Medford, MA'),
('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
('950 Main Street, Worcester, MA 01610');

-- only update the first 3 addresses (323-704 ms - there are caching and shared memory ↵
-- effects so first geocode you do is always slower) --
-- for large numbers of addresses you don't want to update all at once
-- since the whole geocode must commit at once
-- For this example we rejoin with LEFT JOIN
-- and set to rating to -1 rating if no match
-- to ensure we don't regeocode a bad address
UPDATE addresses_to_geocode
  SET (rating, new_address, lon, lat)
    = ( COALESCE(g.rating, -1), pprint_addy(g.addy),
        ST_X(g.geomout)::numeric(8,5), ST_Y(g.geomout)::numeric(8,5) )
FROM (SELECT addid, address
      FROM addresses_to_geocode
      WHERE rating IS NULL ORDER BY addid LIMIT 3) As a
  LEFT JOIN LATERAL geocode(a.address,1) As g ON true
WHERE a.addid = addresses_to_geocode.addid;

result
-----
Query returned successfully: 3 rows affected, 480 ms execution time.

SELECT * FROM addresses_to_geocode WHERE rating is not null;

addid |                address                | lon | lat | ↵
-----+-----+-----+-----+-----
1 | 529 Main Street, Boston MA, 02129 | -71.07177 | 42.38357 | 529 Main St, ↵
  | Boston, MA 02129 | 0 | | |
2 | 77 Massachusetts Avenue, Cambridge, MA 02139 | -71.09396 | 42.35961 | 77 ↵
  | Massachusetts Ave, Cambridge, MA 02139 | 0 | | |
3 | 25 Wizard of Oz, Walaford, KS 99912323 | -97.92913 | 38.12717 | Willowbrook, ↵
  | KS 67502 | 108 | | |
(3 rows)

```

☒☒: ☒☒☒☒☒☒

```

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



Normalize Address, Pprint Addy, ST AsText, ST SnapToGrid, ST X, ST Y

12.2.5 Geocode_Intersection

Geocode_Intersection — **geomout**, **normalized_address** (addy) **TIGER** (edge, face, addr) **PostgreSQL** (soundex, levenshtein)

Synopsis

```
setof record geocode_intersection(text roadway1, text roadway2, text in_state, text in_city, text
in_zip, integer max_results=10, norm addy OUT addy, geometry OUT geomout, integer OUT rating);
```



Normalized address (addy), NAD83 geomout, (rating) PostgreSQL (soundex, levenshtein)

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

数据库 (MA) 使用 TIGER 数据 PostgreSQL 9.0/PostGIS 1.5 数据库 3.0 GHZ 内存 2GB 硬盘 7 分区。数据库包含 3,000 条数据。

TIGER 2011 PostGIS 2.0 PostgreSQL 64 8GB 2003 64
 (41 ).

```
SELECT pprint_addy(addy), st_astext(geomout),rating
      FROM geocode_intersection( 'Haverford St','Germania St', 'MA', 'Boston', '02130',1);
```

pprint_addy	st_astext	rating
98 Haverford St, Boston, MA 02130	POINT(-71.101375 42.31376)	0

64 7 3,500 , 2003 741 .

```
SELECT pprint_addy(addy), st_astext(geomout),rating
      FROM geocode_intersection('Weld', 'School', 'MA', 'Boston');
-----+-----+-----
```

pprint_addy	st_astext	rating
98 Weld Ave, Boston, MA 02119	POINT(-71.099 42.314234)	3
99 Weld Ave, Boston, MA 02119	POINT(-71.099 42.314234)	3

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.

name	setting	unit	category		short_desc
debug_geocode_address	false	boolean	debug		outputs debug information in notice log such as queries when geocode_address is called if true
debug_geocode_intersection	false	boolean	debug		outputs debug information in notice log such as queries when geocode_intersection is called if true
debug_normalize_address	false	boolean	debug		outputs debug information in notice log such as queries and intermediate expressions when normalize_address is called if true
debug_reverse_geocode	false	boolean	debug		if true, outputs debug information in notice log such as queries and intermediate expressions when reverse_geocode
reverse_geocode_numbered_roads	0	integer	rating		For state and county highways, 0 - no preference in name, 1 - prefer the numbered highway name, 2 - prefer local state/county name
use_pagc_address_parser	false	boolean	normalize		If set to true, will try to use the address_standardizer extension (via pagc_normalize_address) instead of tiger normalize_address built one

2.2.0 geocode_settings_default. geocode_settings, geocode_settings

2.1.0

```
SELECT get_geocode_setting('debug_geocode_address') As result;
result
-----
false
```


☐☐

Set_Geocode_Setting

12.2.7 Get_Tract

Get Tract — 返回指定地理坐标 (tract) 所在行政区名称 (field) 的文本值。返回的行政区名称是 NAD83 坐标系下的名称。

Synopsis

```
text get_tract(geometry loc_geom, text output_field=name);
```

☐☐

返回的行政区名称是 NAD83 坐标系下的名称。如果需要使用其他坐标系，可以使用 `ST_Transform` 函数进行转换。

Note

This function uses the census tract which is not loaded by default. If you have already loaded your state table, you can load tract as well as bg, and tabblock using the **Loader_Generate_Census_Script** script.



If you have not loaded your state data yet and want these additional tables loaded, do the following

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

then they will be included by the **Loader_Generate_Script**.

2.0.0 版本开始支持。

☐☐☐☐

```
SELECT get_tract(ST_Point(-71.101375, 42.31376) ) As tract_name;
tract_name
-----
1203.01
```

```
--this one returns the tiger geoid
SELECT get_tract(ST_Point(-71.101375, 42.31376), 'tract_id' ) As tract_id;
tract_id
-----
25025120301
```

☐☐

Geocode>

1. `loader_variables` - 设置变量, 例如, `loader_variables` (staging) 数据库名称。
2. `loader_platform` - 设置平台名称。例如, `loader_platform` 设置为 `linux`。
3. `loader_lookuptables` - 设置查找表 (州, 县), 例如, `loader_lookuptables` 设置为 `usa`。设置 `loader_lookuptables` 为 `usa` 时, `TIGER` 数据库名称为 `tiger`。设置 `loader_lookuptables` 为 `usa` 时, `TIGER` 数据库名称为 `tiger`。设置 `loader_lookuptables` 为 `usa` 时, `TIGER` 数据库名称为 `tiger`。

2.0.0 安装与使用。



Note

Loader_Generate_Script 安装与使用, PostGIS 2.0.0 alpha5 安装与使用 TIGER 数据库名称, 设置 `loader_variables` 为 `usa` 时, `TIGER` 数据库名称为 `tiger`。设置 `loader_variables` 为 `usa` 时, `TIGER` 数据库名称为 `tiger`。

安装与使用

安装与使用。

```
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 安装与使用.
```

```
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
:
:
```

安装与使用

Loader_Generate_Script

12.2.10 Loader_Generate_Script

Loader_Generate_Script — 安装与使用 (州) 安装, TIGER 安装与使用 tiger_data 安装与使用。 (州) 安装与使用。 TIGER 2010 安装与使用, 安装与使用, 安装与使用, 安装与使用。

Synopsis

setof text **loader_generate_script**(text[] param_states, text os);

安装与使用

安装与使用 (州) 安装, TIGER 安装与使用 tiger_data 安装与使用。 (州) 安装与使用。

安装与使用 unzip (安装与使用 7-zip) 安装与使用, 安装与使用 wget 安装与使用。 安装与使用 Section 4.7.2 安装与使用。 安装与使用 (州) 安装与使用。 安装与使用 "staging" "temp" 安装与使用。

安装与使用 OS 安装与使用。

1. loader_variables - 安装与使用, 安装与使用, 安装与使用 (staging) 安装与使用。

**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**

如果您使用 tiger_2010 数据集 (州) 与 tiger_2011 数据集, 请运行 **Drop_Nation_Tables_Generate_Script** 脚本。

要运行

脚本, 请运行以下 SQL 语句:

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

或运行以下 SQL 语句:

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

要运行

Loader_Generate_Script, Missing_Indexes_Generate_Script

12.2.12 Missing_Indexes_Generate_Script

Missing_Indexes_Generate_Script — 生成 (join) 键 (key) 的缺失索引的 SQL DDL 脚本。

Synopsis

text **Missing_Indexes_Generate_Script**();

要运行

tiger 与 **tiger_data** 数据集 (join) 键 (key) 的缺失索引的 SQL DDL 脚本。该脚本将生成缺失索引的 SQL DDL 脚本, 并将其写入到 **ddl** 目录中。该脚本将生成缺失索引的 SQL DDL 脚本, 并将其写入到 **ddl** 目录中。该脚本将生成缺失索引的 SQL DDL 脚本, 并将其写入到 **ddl** 目录中。

2.0.0 版本中已弃用。


```
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 .

11

`norm_addy`.
.

```

tiger tiger_geocoder page_*
TIGER tiger

```

tiger

```

XXXXXXXX norm_addy XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX. XX () XXXXXXXXXXXXXXXXXXXX
XXXXXXXX, [] XXXXXXXXXXXXXXXXXXXX:

```

Normalize Address

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.



This method needs address standardizer extension.

```
(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [state-Abbrev] [zip]
```

[illegible]

house num, predir, name, suftype, sufdir, unit, city, state, postcode

Enhanced: 2.4.0 norm addy object includes additional fields zip4 and address alphanumeric.

1. `address` `TEXT`: `TEXT`.
2. `predirAbbrev` `VARCHAR`: N, S, E, W `TEXT`. `direction_lookup` `TEXT`.
3. `streetName` `VARCHAR`.
4. `streetTypeAbbrev` `VARCHAR`, St, Ave, Cir `TEXT`. `street_type_lookup` `TEXT`.
5. `postdirAbbrev` `VARCHAR`, N, S, E, W `TEXT`. `direction_lookup` `TEXT`.
6. `internal` `VARCHAR`. `TEXT`.
7. `location` `VARCHAR`, `TEXT`.
8. `stateAbbrev` `VARCHAR`, MA, NY, MI `TEXT` (州名) `TEXT`. `state_lookup` `TEXT`.
9. `zip` `VARCHAR`. 02109 `TEXT`.
10. `parsed` `BOOLEAN`. `TEXT`. `normalize_address` `TEXT`.
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 **Pgqz Normalize Address** function. Availability: PostGIS 2.4.0.

address	predirabbrev	streetname	streettypeabbrev	postdirabbrev	internal	
location	stateabbrev	zip	parsed			
9000	E	R00	ST		SUITE 999	
SPRINGFIELD	CO		t			

```
WITH g AS (SELECT address, ROW((sa).house_num, (sa).predir, (sa).name
, (sa).suftype, (sa).sufdir, (sa).unit , (sa).city, (sa).state, (sa).postcode, true)::
norm_addy As na
FROM (SELECT address, standardize_address('tiger.pagc_lex'
, 'tiger.pagc_gaz'
, 'tiger.pagc_rules', address) As sa
FROM addresses_to_geocode) As g)
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
FROM g;
```

orig	streetname	streettypeabbrev
529 Main Street, Boston MA, 02129	MAIN	ST
77 Massachusetts Avenue, Cambridge, MA 02139	MASSACHUSETTS	AVE
25 Wizard of Oz, Walford, 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

```
varchar pprint_addy(norm addy in addy);
```

¶

`norm_addy` 函数, 将地址规范化为标准的格式。 该函数返回一个文本字符串。

¶ `Normalize_Address` 函数。

¶

¶

```
SELECT pprint_addy(normalize_address('202 East Fremont Street, Las Vegas, Nevada 89101'))
  As pretty_address;
      pretty_address
-----
202 E Fremont St, Las Vegas, NV 89101
```

¶

```
SELECT address As orig, pprint_addy(normalize_address(address)) As pretty_address
  FROM addresses_to_geocode;
```

orig	pretty_address
529 Main Street, Boston MA, 02129	529 Main St, Boston MA, 02129
77 Massachusetts Avenue, Cambridge, MA 02139	77 Massachusetts Ave, Cambridge, MA 02139
28 Capen Street, Medford, MA	28 Capen St, Medford, MA
124 Mount Auburn St, Cambridge, Massachusetts 02138	124 Mount Auburn St, Cambridge, MA 02138
950 Main Street, Worcester, MA 01610	950 Main St, Worcester, MA 01610

¶

`Normalize_Address`

12.2.16 Reverse_Geocode

`Reverse_Geocode` — 根据给定的点坐标, 返回该点所在的地址。 该函数返回一个文本字符串。 `include_strnum_range = true` 时, 返回的地址包含门牌号和范围。

Synopsis

record **Reverse_Geocode**(geometry pt, boolean include_strnum_range=false, geometry[] OUT intpt, norm_addy[] OUT addy, varchar[] OUT street);

¶

该函数返回一个记录, 包含以下信息: 点坐标 (geometry), 地址 (varchar), 门牌号和范围 (boolean)。 该函数返回一个记录, 包含以下信息: 点坐标 (geometry), 地址 (varchar), 门牌号和范围 (boolean)。

```

#include <strnum_range.h>.

```

-----	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		↔	↔
cross2							cross1	
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**

安装 TIGER 数据到 PostGIS 数据库。请参考 [Chapter 9 Section 2.2.3](#)。安装 TIGER 数据时，需要指定安装路径，请参考 [Chapter 9 Section 2.2.3](#)。安装 TIGER 数据时，需要指定安装路径，请参考 [Chapter 9 Section 2.2.3](#)。

**Note**

安装 TIGER 数据时，需要指定安装路径，请参考 [Chapter 9 Section 2.2.3](#)。安装 TIGER 数据时，需要指定安装路径，请参考 [Chapter 9 Section 2.2.3](#)。

安装:

1. topo_name - 安装 TIGER 数据到 PostGIS 数据库。
2. region_type - 安装 TIGER 数据时，需要指定安装路径，请参考 [Chapter 9 Section 2.2.3](#)。安装 TIGER 数据时，需要指定安装路径，请参考 [Chapter 9 Section 2.2.3](#)。
3. region_id - TIGER 数据 ID(geoid) 安装 TIGER 数据时，需要指定安装路径，请参考 [Chapter 9 Section 2.2.3](#)。安装 TIGER 数据时，需要指定安装路径，请参考 [Chapter 9 Section 2.2.3](#)。

2.0.0 安装 TIGER 数据。

安装: 安装 TIGER 数据。

安装 TIGER 数据 (2249) 安装 TIGER 数据时，需要指定安装路径，请参考 [Chapter 9 Section 2.2.3](#)。安装 TIGER 数据时，需要指定安装路径，请参考 [Chapter 9 Section 2.2.3](#)。

```
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
-----+-----+-----
```

例: Suffolk County

例: (26986) 0.25, TIGER, , .

```
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
```

例

CreateTopology, CreateTopoGeom, TopologySummary, ValidateTopology

12.2.18 Set_Geocode_Setting

Set_Geocode_Setting —

Synopsis

text Set_Geocode_Setting(text setting_name, text setting_value);

例

tiger.geocode_settings . Get_Geocode_Setting

2.1.0

注意: 在 3.6.0alpha1 版本中

在 3.6.0alpha1 版本中 **Geocode** 函数被弃用, NOTICE 消息将被显示。

```
SELECT set_geocode_setting('debug_geocode_address', 'true') As result;  
result  
-----  
true
```

注意

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_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** - 返回由给定的点集组成的线。
- **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** - 检查两个几何体是否具有相同的对齐方式 (即它们是否位于相同的轴向上)。
- **ST_Union** - Computes a geometry representing the point-set union of the input geometries.
- **ST_Union** - 返回由给定的点集组成的线。
- **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_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** - 3D 几何体的面积。
- **CG_3DDifference** - 3D 几何体的差集。
- **CG_3DIntersection** - 3D 几何体的交集。
- **CG_3DUnion** - Perform 3D union using `postgis_sfcgal`.
- **CG_Volume** - 3D 几何体的体积。
- **ST_3DArea** - 3D 几何体的面积。
- **ST_3DDWithin** - Tests if two 3D geometries are within a given 3D distance
- **ST_3DDifference** - 3D 几何体的差集。

- **ST_3DDistance** - 返回两个几何体 (SRS 相同) 3 维欧几里德距离。
- **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** - 返回几何体的 3 维长度。
- **ST_3DPerimeter** - 返回几何体的 3 维周长。
- **ST_3DUnion** - Perform 3D union.
- **ST_AddEdgeModFace** - 添加边到面, 返回修改后的面, 返回面 ID。
- **ST_AddEdgeNewFaces** - 添加边到面, 返回新的面, 返回面 ID 2 个值。
- **ST_AddIsoEdge** - 添加孤立边 anode 和 another node 之间的边 alinestring 返回 ID 值。
- **ST_AddIsoNode** - 添加孤立节点 (isolated) 返回 ID 值。返回 NULL 值, 表示失败。
- **ST_Area** - 返回几何体的面积。
- **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 表示的几何体。
- **ST_AsText** - 返回 WKT(Well-Known Text) 表示的几何体 SRID 值。
- **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** - 返回几何体的质心。
- **ST_ChangeEdgeGeom** - 修改几何体的边。
- **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 的坐标维度。
- **ST_CreateTopoGeo** - 创建拓扑几何体。
- **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 的维度。
- **ST_Disjoint** - Tests if two geometries have no points in common
- **ST_Distance** - 返回两个几何体 3 维欧几里德 (longest) 距离。

- **ST_EndPoint** - ST_LineString 或 ST_CircularString 的末端点。
- **ST_Envelope** - 给定几何体的包络线 (double precision; float8) 的几何体。
- **ST_Equals** - Tests if two geometries include the same set of points
- **ST_ExteriorRing** - 给定几何体的外部环。
- **ST_GMLToSQL** - GML 几何体到 ST_Geometry 的转换。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 几何体到 ST_Geometry 的转换。
- **ST_GeomFromWKB** - WKB(Well-Known Binary) 几何体到 SRID 的转换。
- **ST_GeometryFromText** - WKT(Well-Known Text) 几何体到 ST_Geometry 的转换。或 ST_GeomFromText 的别名。
- **ST_GeometryN** - ST_Geometry 的 N 维子几何体。
- **ST_GeometryType** - ST_Geometry 的几何体类型。
- **ST_GetFaceEdges** - aface 的边。
- **ST_GetFaceGeometry** - 给定 ID 的几何体。
- **ST_InitTopoGeo** - Creates a new topology schema and registers it in the topology.topology table.
- **ST_InteriorRingN** - 给定几何体的 N 维内部环。
- **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 表示是，FALSE 表示否。
- **ST_IsEmpty** - Tests if a geometry is empty.
- **ST_IsRing** - Tests if a LineString is closed and simple.
- **ST_IsSimple** - 几何体是否是简单的。TRUE 表示是，FALSE 表示否。
- **ST_IsValid** - Tests if a geometry is well-formed in 2D.
- **ST_Length** - 几何体的长度。
- **ST_LineFromText** - 给定 SRID 的 WKT 几何体到 ST_Geometry 的转换。SRID 为 0 时默认为 0。
- **ST_LineFromWKB** - 给定 SRID 的 WKB 几何体到 LINESTRING 的转换。
- **ST_LinestringFromWKB** - 给定 SRID 的 WKB 几何体到 LINESTRING 的转换。
- **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 几何体到 ST_MultiLineString 的转换。

- **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** - 返回修改后的边 ID，返回修改后的边 ID。返回修改后的边 ID。
- **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** - 返回修改后的边 ID，返回修改后的边 ID 2 返回修改后的边 ID。返回修改后的边 ID ID 返回修改后的边 ID。
- **ST_NumCurves** - Return the number of component curves in a CompoundCurve.
- **ST_NumGeometries** - 返回几何体的数量。返回几何体的数量。
- **ST_NumInteriorRings** - 返回内部环的数量。
- **ST_NumPatches** - 返回补丁的数量。返回补丁的数量 NULL 返回补丁的数量。
- **ST_NumPoints** - ST_LineString 返回 ST_CircularString 返回补丁的数量。
- **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 返回补丁的数量。
- **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** - 返回 SRID 返回 WKT 返回补丁的数量。SRID 返回补丁的数量 返回补丁的数量 0 返回补丁的数量。
- **ST_PointFromWKB** - 返回 SRID 返回 WKB 返回补丁的数量。
- **ST_PointN** - ST_LineString 返回 ST_CircularString 返回补丁的数量。
- **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** - 返回修改后的边 ID，返回修改后的边 ID 返回修改后的边 ID 返回修改后的边 ID。

- **ST_RemoveIsoEdge** - Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown.
- **ST_RemoveIsoNode** - 移除几何中的孤立节点并返回描述。如果节点不是孤立的，则抛出异常。(移除孤立的节点) 返回, 移除孤立的节点。
- **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 维几何体的体积。返回几何体 (3 维几何体) 0 维几何体的体积。
- **ST_WKBToSQL** - WKB(Well-Known Binary) 格式化的 ST_Geometry 格式化的。返回 SRID 格式化的 ST_GeomFromWKB 格式化的。
- **ST_WKTToSQL** - WKT(Well-Known Text) 格式化的 ST_Geometry 格式化的。返回 ST_GeomFromWKT 格式化的。
- **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** - 返回几何体的面积。
- **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) 格式化的 SRID 格式化的。
- **ST_AsGML** - 返回 GML 2 或 GML 3 格式化的。

- **ST_AsGeoJSON** - Return a geometry or feature in GeoJSON format.
- **ST_AsKML** - 将 GML 2 转换为 GML 3 格式。
- **ST_AsSVG** - Returns SVG path data for a geometry.
- **ST_AsText** - 将 WKT(Well-Known Text) 转换为 SRID 格式。
- **ST_Azimuth** - 计算 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** - 计算几何体的质心。
- **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** - 计算 3 个几何体 (longest) 之间的距离。
- **ST_GeogFromText** - WKT (几何体) 转换为 Geography 格式。
- **ST_GeogFromWKB** - WKB 格式转换为 EWKB(二进制 WKB) 格式。
- **ST_GeographyFromText** - WKT (几何体) 转换为 Geography 格式。
- **=** - 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** - 计算几何体的长度。
- **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** - 计算 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 (TRUE 表示重合)。

13.5 PostGIS Raster Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a **raster** data type object. Listed in alphabetical order.

- **Box3D** - 返回 BOX3D 几何体。
- **@** - B 与 A 相等返回 TRUE。不相等返回 FALSE。
- **~** - A 与 B 相等返回 TRUE。不相等返回 FALSE。
- **=** - A 与 B 相等返回 TRUE。不相等返回 FALSE。
- **&&** - A 与 B 相等返回 TRUE。不相等返回 FALSE。
- **&<** - A 与 B 相等返回 TRUE。不相等返回 FALSE。
- **&>** - A 与 B 相等返回 TRUE。不相等返回 FALSE。
- **~=** - A 与 B 相等返回 TRUE。不相等返回 FALSE。
- **ST_Retile** - 将栅格重新分块，返回新的栅格。
- **ST_AddBand** - 向栅格添加新的波段 (n) 个波段。返回新的栅格。
- **ST_AsBinary/ST_AsWKB** - Return the Well-Known Binary (WKB) representation of the raster.
- **ST_AsGDALRaster** - Return the raster tile in the designated GDAL Raster format. Raster formats are one of those supported by your compiled library. Use ST_GDALDrivers() to get a list of formats supported by your library.
- **ST_AsHexWKB** - Return the Well-Known Binary (WKB) in Hex representation of the raster.
- **ST_AsJPEG** - 将栅格转换为 JPEG (Joint Photographic Experts Group) 格式 (8 位) 的字节数组。返回 1 个字节数组 3 个字节数组 RGB 格式。
- **ST_AsPNG** - 将栅格转换为 PNG (Portable Network Graphics) 格式 (8 位) 的字节数组。返回 1 个字节数组 3 个字节数组 4 个字节数组 RGB 格式。返回 2 个字节数组 4 个字节数组 RGB 格式。返回 1 个字节数组。返回 RGB 或 RGBA 格式。
- **ST_AsRaster** - PostGIS 栅格 PostGIS 栅格。
- **ST_AsRasterAgg** - Aggregate. Renders PostGIS geometries into a new raster.
- **ST_AsTIFF** - Return the raster selected bands as a single TIFF image (byte array). If no band is specified or any of specified bands does not exist in the raster, then will try to use all bands.
- **ST_Aspect** - 返回 (度) 方位角。返回。
- **ST_Band** - 返回栅格的波段。返回。
- **ST_BandFileSize** - Returns the file size of a band stored in file system. If no bandnum specified, 1 is assumed.
- **ST_BandFileTimestamp** - Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.
- **ST_BandIsNoData** - 返回 NODATA 几何体。
- **ST_BandMetaData** - 返回栅格的波段元数据。返回 1 个字节数组。

- **ST_BandNoDataValue** - 返回栅格的 NODATA 值。如果栅格没有 NODATA 值，则返回 1。
- **ST_BandPath** - 返回栅格的波段路径。bandnum 指定波段编号，从 1 开始。
- **ST_BandPixelType** - 返回栅格的波段像素类型。bandnum 指定波段编号，从 1 开始。
- **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** - 返回栅格的 8BUI 颜色映射 (grayscale, RGB, RGBA) 的 4 个波段。如果指定了 1，则返回 1 个波段。
- **ST_Contains** - 返回栅格 rastA 是否包含栅格 rastB。如果 rastB 包含 rastA，则返回 TRUE。
- **ST_ContainsProperly** - rastB 是否包含 rastA。如果 rastA 包含 rastB，则返回 TRUE。
- **ST_Contour** - Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.
- **ST_ConvexHull** - BandNoDataValue 返回栅格的 NODATA 值，如果栅格没有 NODATA 值，则返回 1。ST_Envelope 返回栅格的包络线。
- **ST_Count** - 返回栅格中指定值的像素数量。exclude_nodata_value 指定是否排除 NODATA 值。
- **ST_CountAgg** - 返回栅格中指定值的像素数量。exclude_nodata_value 指定是否排除 NODATA 值。
- **ST_CoveredBy** - 返回栅格 rastA 是否被栅格 rastB 覆盖。
- **ST_Covers** - 返回栅格 rastB 是否覆盖栅格 rastA。
- **ST_DFullyWithin** - 返回栅格 rastA 是否完全在栅格 rastB 的指定距离内。
- **ST_DWithin** - 返回栅格 rastA 是否在栅格 rastB 的指定距离内。
- **ST_Disjoint** - 返回栅格 rastA 是否与栅格 rastB 不相交。
- **ST_DumpAsPolygons** - 返回栅格 geomval(geom, val) 的几何值。如果指定了 1，则返回 1 个几何值。
- **ST_DumpValues** - 返回栅格的 2 个波段值。
- **ST_Envelope** - 返回栅格的包络线。
- **ST_FromGDALRaster** - 从 GDAL 栅格创建 PostGIS 栅格。
- **ST_GeoReference** - 返回 (world) 返回栅格的地理参考信息。GDAL 返回 ESRI 的地理参考信息。如果指定了 GDAL，则返回 GDAL 的地理参考信息。
- **ST_Grayscale** - Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue
- **ST_HasNoBand** - 返回栅格是否有指定的波段。如果指定了 1，则返回 1。

- **ST_Height** - 返回栅格的最大高度值。
- **ST_HillShade** - 根据栅格数据生成地形阴影图。参数包括：栅格、偏移量、旋转角度、颜色方案等。
- **ST_Histogram** - 返回栅格的直方图。参数包括：bin（直方图宽度）、栅格、是否归一化等。
- **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** - 返回两个栅格的交集。参数包括：栅格A、栅格B。
- **ST_Intersects** - 返回两个栅格是否相交。参数包括：rastA、rastB。
- **ST_IsEmpty** - 返回栅格是否为空（width = 0, height = 0）。参数包括：栅格。
- **ST_MakeEmptyCoverage** - Cover georeferenced area with a grid of empty raster tiles.
- **ST_MakeEmptyRaster** - 返回空栅格。参数包括：宽度、高度、原点X、原点Y、比例尺（scalex, scaley, skewx & skewy）、SRID（空间参考系统ID）等。
- **ST_MapAlgebra (callback function version)** - 返回栅格。参数包括：1 个回调函数、栅格A、栅格B。
- **ST_MapAlgebraExpr** - 返回栅格。参数包括：1 个表达式、栅格A、栅格B。表达式可以是 PostgreSQL 表达式。
- **ST_MapAlgebraExpr** - 返回栅格。参数包括：2 个表达式、栅格A、栅格B。表达式可以是 PostgreSQL 表达式。
- **ST_MapAlgebraFct** - 返回栅格。参数包括：1 个函数名、栅格A、栅格B。函数名可以是 PostgreSQL 函数名。
- **ST_MapAlgebraFct** - 返回栅格。参数包括：2 个函数名、栅格A、栅格B。函数名可以是 PostgreSQL 函数名。
- **ST_MapAlgebraFctNgb** - 返回栅格。参数包括：1 个函数名、栅格A、栅格B。函数名可以是 PostgreSQL 函数名。
- **ST_MapAlgebra (expression version)** - 返回栅格。参数包括：1 个表达式、栅格A、栅格B。表达式可以是 PostgreSQL 表达式。
- **ST_MemSize** - 返回栅格的内存大小。参数包括：栅格。
- **ST_MetaData** - 返回栅格的元数据。参数包括：栅格、偏移量、旋转角度等。
- **ST_MinConvexHull** - 返回栅格的最小凸包。参数包括：栅格。
- **ST_NearestValue** - columnx 行号，返回最近的值。参数包括：栅格、NODATA 值。

- **ST_Neighborhood** - columnx 和 rowy, 返回指定像素邻域中的像素值。如果指定了 NODATA 则返回 NODATA。2 个参数返回 2 个像素值。
- **ST_NotSameAlignmentReason** - 返回两个栅格的不匹配原因。
- **ST_NumBands** - 返回栅格的波段数。
- **ST_Overlaps** - 返回两个栅格是否重叠。
- **ST_PixelAsCentroid** - 返回指定像素的质心 (X, Y)。
- **ST_PixelAsCentroids** - 返回指定像素的质心 (X, Y)。
- **ST_PixelAsPoint** - 返回指定像素的点。
- **ST_PixelAsPoints** - 返回指定像素的点 (X, Y)。
- **ST_PixelAsPolygon** - 返回指定像素的多边形。
- **ST_PixelAsPolygons** - 返回指定像素的多边形 (X, Y)。
- **ST_PixelHeight** - 返回指定像素的高度。
- **ST_PixelOfValue** - 返回指定像素的 columnx, rowy。
- **ST_PixelWidth** - 返回指定像素的宽度。
- **ST_Polygon** - NODATA 返回 NODATA。
- **ST_Quantile** - 返回指定像素的 (population) 分位数 (quantile) 值。例如, 返回 25%, 50%, 75% 分位数 (percentile) 值。
- **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** - 返回指定像素的 X, Y (X, Y) 坐标。
- **ST_RasterToWorldCoordX** - 返回指定像素的 X 坐标。
- **ST_RasterToWorldCoordY** - 返回指定像素的 Y 坐标。
- **ST_Reclass** - 返回指定像素的重新分类后的值。nband 返回 nband 个值。nband 返回 1 个值。例如: 16BUI 返回 8BUI 值。
- **ST_Resample** - 返回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** - 返回指定像素的重新大小后的值。

- **ST_Reskew** - 返回 (列向量) 最近邻插值。 NearestNeighbor(列向量), Bilinear, Cubic, CubicSpline 或 Lanczos 最近邻插值。 NearestNeighbor 列向量。
- **ST_Rotation** - 返回旋转后的列向量。
- **ST_Roughness** - DEM 列向量" 粗糙度 (roughness)" 列向量。
- **ST_SRID** - spatial_ref_sys 列向量, 返回列向量。
- **ST_SameAlignment** - 返回列向量, 列, 列向量, 列向量 (列向量) 列向量, 列向量。
- **ST_ScaleX** - 返回列向量 X 列向量。
- **ST_ScaleY** - 返回列向量 Y 列向量。
- **ST_SetBandIndex** - Update the external band number of an out-db band
- **ST_SetBandIsNoData** - 列 isnodata 列。
- **ST_SetBandNoDataValue** - NODATA 列。 列 1 列。 列 NODATA 列, nodata value = NULL 列。
- **ST_SetBandPath** - Update the external path and band number of an out-db band
- **ST_SetGeoReference** - 列 6 列。 列。 GDAL 列 ESRI 列。 列 GDAL 列。
- **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** - 列。
- **ST_SetSRID** - 列 SRID 列 spatial_ref_sys 列 SRID 列。
- **ST_SetScale** - X 列 Y 列。 列/列/列。
- **ST_SetSkew** - 列 X 列 Y 列 (skew)(列) 列。 列, X 列 Y 列。
- **ST_SetUpperLeft** - Sets the value of the upper left corner of the pixel of the raster to projected X and Y coordinates.
- **ST_SetValue** - 列 columnx, rowy 列。 列 1 列, 列 1 列。
- **ST_SetValues** - 列。
- **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** - 列 X 列 (skew)(列) 列。
- **ST_SkewY** - 列 Y 列 (skew)(列) 列。
- **ST_Slope** - 列 (列) 列。 列。
- **ST_SnapToGrid** - 列。 NearestNeighbor(列), Bilinear, Cubic, CubicSpline 或 Lanczos 最近邻插值。 NearestNeighbor 列。
- **ST_Summary** - 列。

- **ST_SummaryStats** - 返回栅格的统计信息 (count, sum, mean, stddev, min, max) 以及栅格的范围。返回 1 个结果。
- **ST_SummaryStatsAgg** - 聚合栅格的统计信息 (count, sum, mean, stddev, min, max) 以及栅格的范围。返回 1 个结果。
- **ST_TPI** - 地形位置指数 (Topographic Position Index) 栅格。
- **ST_TRI** - 地形崎岖度指数 (Terrain Ruggedness Index) 栅格。
- **ST_Tile** - 将栅格分割成指定大小的瓦片。
- **ST_Touches** - 检查两个栅格 (rastA, rastB) 是否接触。TRUE 表示接触。
- **ST_Transform** - 将栅格从一种坐标系转换到另一种坐标系。NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos 是可选的插值方法。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** - 返回栅格在指定位置 (columnx, rowy) 的值。exclude_nodata_value 参数用于处理无数据值。
- **ST_ValueCount** - 返回栅格中每个值的出现次数 (value, count)。NODATA 值被排除在外。
- **ST_Width** - 返回栅格的宽度。
- **ST_Within** - 检查一个栅格 (rastB) 是否完全包含在另一个栅格 (rastA) 内部。
- **ST_WorldToRasterCoord** - 将世界坐标 (X, Y) 转换为栅格坐标 (xw, yw)。
- **ST_WorldToRasterCoordX** - 将世界坐标 (pt) 转换为栅格 X 坐标 (xw, yw)。
- **ST_WorldToRasterCoordY** - 将世界坐标 (pt) 转换为栅格 Y 坐标 (xw, yw)。
- **UpdateRasterSRID** - 更新栅格的 SRID。

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** - 将 geomval(geom, val) 转换为多边形集合。返回 1 个结果。
- **ST_Intersection** - 返回两个几何体的交集。

- **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。

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** - 返回几何体中所有BOX3D的集合。
- **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** - 返回TWKB(Tiny Well-Known Binary)格式的几何体。
- **ST_Box2dFromGeoHash** - GeoHash 返回BOX2D。
- **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 bounding box.
- `~(geometry,box2df)` - Returns TRUE if a geometry's 2D bounding box contains a 2D float precision bounding box (BOX2DF).
- `@(box2df,box2df)` - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- `@(box2df,geometry)` - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- `@(geometry,box2df)` - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- `&&(box2df,box2df)` - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- `&&(box2df,geometry)` - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- `&&(geometry,box2df)` - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

13.8 PostGIS Functions that support 3D

The functions given below are PostGIS functions that do not throw away the Z-Index.

- **AddGeometryColumn** - `varchar(255), geometry, srid, nullable, use_index`.
- **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
- **CG_3DAlphaWrapping** - Computes a 3D Alpha-wrapping strictly enclosing a geometry.
- **CG_3DArea** - 3 `double`. `0` `double`.
- **CG_3DConvexHull** - `geometry`.
- **CG_3DDifference** - 3 `geometry`.
- **CG_3DIntersection** - 3 `geometry`.
- **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** - `geometry`.
- **CG_ConstrainedDelaunayTriangles** - Return a constrained Delaunay triangulation around the given input geometry.
- **CG_Extrude** - `geometry`.
- **CG_ForceLHR** - LHR(Left Hand Reverse; `boolean`) `geometry`.

- **CG_IsPlanar** - 测试几何体是否是平面。
- **CG_IsSolid** - 测试几何体是否是实心体。返回 true 表示是实心体，false 表示不是。
- **CG_MakeSolid** - 将几何体转换为实心体。如果输入是面，则返回由该面定义的实心体。如果输入是点，则返回由该点定义的实心体。如果输入是线，则返回由该线定义的实心体。如果输入是 TIN，则返回由该 TIN 定义的实心体。
- **CG_Orientation** - 返回几何体的朝向 (orientation)。
- **CG_RotateX** - 围绕 X 轴旋转几何体。
- **CG_RotateY** - 围绕 Y 轴旋转几何体。
- **CG_RotateZ** - 围绕 Z 轴旋转几何体。
- **CG_Simplify** - 减少几何体的复杂度，同时保留基本特征和 Z/M 值。
- **CG_StraightSkeleton** - 计算几何体的 (straight skeleton)。
- **CG_Tessellate** - 对几何体进行 (tessellation) 操作。返回 TIN 或 TIN 列表。
- **CG_Visibility** - 计算从一个点或一个段在几何体中的可见性多边形。
- **CG_Volume** - 计算 3D 几何体的体积。返回 0 表示不是 3D 几何体。
- **DropGeometryColumn** - 从表中删除几何体列。
- **GeometryType** - 返回 ST_Geometry 类型的名称。
- **ST_3DArea** - 计算 3D 几何体的面积。返回 0 表示不是 3D 几何体。
- **ST_3DClosestPoint** - 返回 g2 中距离 g1 最近的点。如果 g2 是 3D 几何体，则返回 3D 点。
- **ST_3DConvexHull** - 计算 3D 几何体的凸包。
- **ST_3DDFullyWithin** - 测试两个 3D 几何体是否完全在一个给定的 3D 距离内。
- **ST_3DDWithin** - 测试两个 3D 几何体是否在一个给定的 3D 距离内。
- **ST_3DDifference** - 计算 3D 几何体的差集。
- **ST_3DDistance** - 计算两个 3D 几何体之间的距离。如果 SRS 是 3D，则返回 3D 距离。
- **ST_3DExtent** - 聚合函数，返回几何体的 3D 边界框。
- **ST_3DIntersection** - 计算 3D 几何体的交集。
- **ST_3DIntersects** - 测试两个几何体是否在 3D 空间中相交。仅适用于点、线串、多边形、多面体表面 (area)。
- **ST_3DLength** - 计算 3D 几何体的长度。
- **ST_3DLineInterpolatePoint** - 返回沿 3D 线在分数位置上的点。
- **ST_3DLongestLine** - 返回 3D 几何体中最长的线。
- **ST_3DMaxDistance** - 计算两个 3D 几何体之间的最大距离。如果 SRS 是 3D，则返回 3D 距离。
- **ST_3DPerimeter** - 计算 3D 几何体的周长。
- **ST_3DShortestLine** - 返回 3D 几何体中最短的线。

- **ST_3DUnion** - Perform 3D union.
- **ST_AddMeasure** - Interpolates measures along a linear geometry.
- **ST_AddPoint** - 向几何体添加点。
- **ST_Affine** - Apply a 3D affine transformation to a geometry.
- **ST_ApproximateMedialAxis** - 近似 medial axis。
- **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 的 EWKT。
- **ST_AsGML** - 将 GML 2 转换为 GML 3。
- **ST_AsGeoJSON** - Return a geometry or feature in GeoJSON format.
- **ST_AsHEXEWKB** - 将 (NDR) 转换为 (XDR) 的 HEXEWKB (二进制) 格式。
- **ST_AsKML** - 将 GML 2 转换为 GML 3。
- **ST_AsX3D** - 将 X3D XML 转换为: ISO-IEC-19776-1.2-X3DEncodings-XML 格式。
- **ST_Boundary** - 返回几何体的边界。
- **ST_BoundingDiagonal** - 返回几何体的包围盒的对角线。
- **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** - 将几何体转换为点。
- **ST_DumpRings** - Returns a set of geometry_dump rows for the exterior and interior rings of a Polygon.

- **ST_DumpSegments** - 将几何体分解为线段并返回。
- **ST_EndPoint** - ST_LineString 或 ST_CircularString 的末端点。
- **ST_ExteriorRing** - 返回多边形的外环。
- **ST_Extrude** - 将 2D 几何体拉伸为 3D 几何体。
- **ST_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
- **ST_Force2D** - 强制将 3D 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) 强制多边形。
- **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** - 强制多边形 (orientation) 遵循右手规则 (Right-Hand Rule)。
- **ST_ForceSFS** - 强制符合 SFS 1.1 规范。
- **ST_Force3D** - 强制 XYZ 坐标。
- **ST_Force3DZ** - 强制 XYZ 坐标，Z 值由函数指定。
- **ST_Force4D** - 强制 XYZM 坐标。
- **ST_ForceCollection** - 强制将几何体集合转换为集合。
- **ST_GeomFromEWKB** - EWKB(Extended Well-Known Binary) 格式的 ST_Geometry。
- **ST_GeomFromEWKT** - EWKT(Extended Well-Known Text) 格式的 ST_Geometry。
- **ST_GeomFromGML** - GML 格式的 PostGIS 几何体。
- **ST_GeomFromGeoJSON** - GeoJSON 格式的 PostGIS 几何体。
- **ST_GeomFromKML** - KML 格式的 PostGIS 几何体。
- **ST_GeometricMedian** - 计算几何体的 (median) 几何体。
- **ST_GeometryN** - ST_Geometry 集合中的第 N 个几何体。
- **ST_GeometryType** - ST_Geometry 的几何类型。
- **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** - 返回多边形的第 N 个内环。
- **ST_InterpolatePoint** - 在几何体上插值 (M 值) 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)。
- **ST_IsCollection** - 是否是几何体集合 (TRUE 或 FALSE)。
- **ST_IsPlanar** - 是否是平面几何体。

- **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** - 返回几何体是否简单。TRUE 表示简单，FALSE 表示不简单。
- **ST_IsSolid** - 返回几何体是否是实心体。返回 TRUE 表示是实心体，FALSE 表示不是。
- **ST_IsValidTrajectory** - Tests if the geometry is a valid trajectory.
- **ST_LengthSpheroid** - 返回几何体在球面上的长度。
- **ST_LineFromMultiPoint** - 从多点集合中生成一条线。
- **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** - 从点集合中生成一条线。
- **ST_MakePoint** - Creates a 2D, 3DZ or 4D Point.
- **ST_MakePolygon** - Creates a Polygon from a shell and optional list of holes.
- **ST_MakeSolid** - 从几何体生成实心体。返回 TRUE 表示是实心体，FALSE 表示不是。支持 TIN 格式。
- **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** - 返回几何体中的点数 (不包括空洞)。
- **ST_NRings** - 返回几何体中的环数。
- **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。
- **ST_NumPatches** - 返回几何体集合中的补丁数量。NULL 表示没有补丁。
- **ST_Orientation** - 返回几何体的朝向 (orientation)。
- **ST_PatchN** - ST_Geometry 的补丁 N。
- **ST_PointFromWKB** - 从 WKB 格式生成点。SRID 和 WKB 格式。
- **ST_PointN** - ST_LineString 或 ST_CircularString 的第 N 个点。
- **ST_PointOnSurface** - Computes a point guaranteed to lie in a polygon, or on a geometry.

- **ST_Points** - 将几何体转换为点集。
- **ST_Polygon** - Creates a Polygon from a LineString with a specified SRID.
- **ST_RemovePoint** - Remove a point from a linestring.
- **ST_RemoveRepeatedPoints** - Returns a version of a geometry with duplicate points removed.
- **ST_Reverse** - 将几何体反向。
- **ST_Rotate** - Rotates a geometry about an origin point.
- **ST_RotateX** - Rotates a geometry about the X axis.
- **ST_RotateY** - Rotates a geometry about the Y axis.
- **ST_RotateZ** - Rotates a geometry about the Z axis.
- **ST_Scale** - Scales a geometry by given factors.
- **ST_Scroll** - Change start point of a closed LineString.
- **ST_SetPoint** - 将点设置到几何体的指定位置。
- **ST_ShiftLongitude** - Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- **ST_SnapToGrid** - 将几何体 snap 到网格。
- **ST_StartPoint** - Returns the first point of a LineString.
- **ST_StraightSkeleton** - 计算 (straight skeleton) 骨架。
- **ST_SwapOrdinates** - 交换几何体的纵坐标和横坐标。
- **ST_SymDifference** - Computes a geometry representing the portions of geometries A and B that do not intersect.
- **ST_Tessellate** - 将几何体 tessellation 为 TIN 三角网。
- **ST_TransScale** - Translates and scales a geometry by given offsets and factors.
- **ST_Translate** - Translates a geometry by given offsets.
- **ST_UnaryUnion** - Computes the union of the components of a single geometry.
- **ST_Union** - Computes a geometry representing the point-set union of the input geometries.
- **ST_Volume** - 3D 几何体的体积 (0 到正无穷)。
- **ST_WrapX** - X 坐标的环绕。
- **ST_X** - Returns the X coordinate of a Point.
- **ST_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- **ST_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST_Y** - Returns the Y coordinate of a Point.
- **ST_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- **ST_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- **ST_Z** - Returns the Z coordinate of a Point.
- **ST_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **ST_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.

- **ST_Zmflag** - ST_Geometry 返回 Z 标志。
- **Equals** - 返回 TopoGeometry 是否相等。
- **Intersects** - 返回 TopoGeometry 是否相交。
- **UpdateGeometrySRID** - Updates the SRID of all features in a geometry column, and the table meta-data.
- **&&&** - A 与 n 维 B 的 n 维 TRUE 相交。
- **&&&(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** - 添加几何列。
- **Box2D** - Returns a BOX2D representing the 2D extent of a geometry.
- **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
- **DropGeometryColumn** - 删除几何列。
- **GeometryType** - ST_Geometry 返回几何类型。
- **PostGIS_AddBBox** - 添加 BBox。
- **PostGIS_DropBBox** - 删除 BBox。
- **PostGIS_HasBBox** - 检查 BBox 是否存在。
- **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** - 返回 WKT(Well-Known Text) 和 SRID 的文本表示。
- **ST_AsHEXEWKB** - 返回 (NDR) 和 (XDR) 的 HEXEWKB (HEX) 表示。
- **ST_AsSVG** - Returns SVG path data for a geometry.
- **ST_AsText** - 返回 WKT(Well-Known Text) 和 SRID 的文本表示。
- **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 返回的维数。
- **ST_CurveToLine** - Converts a geometry containing curves to a linear geometry.
- **ST_Distance** - 返回两个几何体之间的最短 (shortest) 或最长 (longest) 距离。
- **ST_Dump** - Returns a set of geometry_dump rows for the components of a geometry.
- **ST_DumpPoints** - 返回几何体中的点。
- **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** - 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_ForceSFS** - 强制将几何体转换为 SFS 1.1 兼容格式。
- **ST_Force3D** - 强制将几何体转换为 XYZ 格式。ST_Force3DZ 强制为 3DZ。
- **ST_Force3DM** - 强制将几何体转换为 XYM 格式。
- **ST_Force3DZ** - 强制将几何体转换为 XYZ 格式。
- **ST_Force4D** - 强制将几何体转换为 XYZM 格式。
- **ST_ForceCollection** - 强制将几何体转换为集合。
- **ST_GeoHash** - 返回 GeoHash 字符串。
- **ST_GeogFromWKB** - WKB 转换为 EWKB (Extended WKB) 格式。
- **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_GeomFromText** - WKT 转换为 ST_Geometry 格式。
- **ST_GeomFromWKB** - WKB(Well-Known Binary) 转换为 SRID 指定的 ST_Geometry 格式。
- **ST_GeometryN** - ST_Geometry 集合中的第 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 与 B 的交集是否为空。TRUE 表示非空。
- **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 是否是闭合的。TRUE 表示闭合。多边形 (POLYGON) 总是 TRUE。
- **ST_IsCollection** - 是否是 GeometryCollection。TRUE 表示是。

- **ST_IsEmpty** - Tests if a geometry is empty.
- **ST_LineToCurve** - Converts a linear geometry to a curved geometry.
- **ST_MemSize** - ST_Geometry 内存大小。
- **ST_NPoints** - 返回几何体 (点集) 中的点数。
- **ST_NRings** - 返回几何体 (多边形) 中的环数。
- **ST_PointFromWKB** - 将 SRID 和 WKB 数据转换为点。
- **ST_PointN** - ST_LineString 或 ST_CircularString 中的第 N 个点。
- **ST_Points** - 返回几何体中的所有点。
- **ST_Rotate** - 围绕原点旋转几何体。
- **ST_RotateZ** - 围绕 Z 轴旋转几何体。
- **ST_SRID** - 返回几何体的空间参考标识符。
- **ST_Scale** - 按给定因子缩放几何体。
- **ST_SetSRID** - 设置几何体的 SRID。
- **ST_StartPoint** - 返回 LineString 的第一个点。
- **ST_Summary** - 返回几何体的摘要信息。
- **ST_SwapOrdinates** - 交换几何体中坐标的顺序。
- **ST_TransScale** - 按给定偏移量和因子缩放并平移几何体。
- **ST_Transform** - 返回将坐标转换到不同空间参考系的新几何体。
- **ST_Translate** - 按给定偏移量平移几何体。
- **ST_XMax** - 返回 2D 或 3D 包围盒或几何体的 X 最大值。
- **ST_XMin** - 返回 2D 或 3D 包围盒或几何体的 X 最小值。
- **ST_YMax** - 返回 2D 或 3D 包围盒或几何体的 Y 最大值。
- **ST_YMin** - 返回 2D 或 3D 包围盒或几何体的 Y 最小值。
- **ST_ZMax** - 返回 2D 或 3D 包围盒或几何体的 Z 最大值。
- **ST_ZMin** - 返回 2D 或 3D 包围盒或几何体的 Z 最小值。
- **ST_Zmflag** - ST_Geometry 的 Z 标志。
- **UpdateGeometrySRID** - 更新几何体列中所有特征体的 SRID，以及表元数据。
- **~(box2df,box2df)** - 返回 TRUE 如果 2D 浮点精度包围盒 (BOX2DF) 包含另一个 2D 浮点精度包围盒 (BOX2DF)。
- **~(box2df,geometry)** - 返回 TRUE 如果 2D 浮点精度包围盒 (BOX2DF) 包含几何体的 2D 包围盒。
- **~(geometry,box2df)** - 返回 TRUE 如果几何体的 2D 包围盒包含 2D 浮点精度包围盒 (BOX2DF)。
- **&&** - A 是 2D 几何体 B 是 2D 几何体 且 两者都是 TRUE 则为 TRUE。
- **&&&** - A 是 n 维几何体 B 是 n 维几何体 且 两者都是 TRUE 则为 TRUE。

- `@(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 $\times \dots \times \dots$. $\dots$ 0 $\dots$.
- **CG_3DConvexHull** - $\dots$.
- **CG_3DDifference** - 3 $\dots$.
- **CG_3DIntersection** - 3 $\dots$.
- **CG_3DUnion** - Perform 3D union using postgis_sfcgal.
- **CG_ApproximateMedialAxis** - $\dots$.
- **CG_Extrude** - $\dots$.
- **CG_ForceLHR** - LHR(Left Hand Reverse; $\dots$) $\dots$.
- **CG_IsPlanar** - $\dots$.
- **CG_IsSolid** - $\dots$. $\dots$.
- **CG_MakeSolid** - $\dots$. $\dots$. $\dots$, $\dots$ TIN $\dots$.
- **CG_StraightSkeleton** - $\dots$ (straight skeleton) $\dots$.
- **CG_Tessellate** - $\dots$ (tessellation) $\dots$ TIN $\dots$ TIN $\dots$.

- **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。返回非负实数 (0 表示空集)。
- **GeometryType** - ST_Geometry 类型的名称。
- **ST_3DArea** - 3 维几何体的面积。返回非负实数。
- **ST_3DClosestPoint** - g2 几何体到 g1 几何体 3 维最近点的坐标。返回 3D 点几何体。
- **ST_3DConvexHull** - 3 维几何体的凸包。
- **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** - 3 维几何体之间的最小距离 (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** - 3 维几何体中最长的线 (longest) 的坐标。
- **ST_3DMaxDistance** - 3 维几何体之间的最大距离 (SRS 无关) 3 维几何体之间的最大距离。
- **ST_3DShortestLine** - 3 维几何体中最短的线 (shortest) 的坐标。
- **ST_3DUnion** - Perform 3D union.
- **ST_Affine** - Apply a 3D affine transformation to a geometry.
- **ST_ApproximateMedialAxis** - 3 维几何体的近似中轴。
- **ST_Area** - 3 维几何体的面积。
- **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** - 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_AsX3D** - 返回 X3D XML 格式的几何体: ISO-IEC-19776-1.2-X3DEncodings-XML 格式。
- **ST_CoordDim** - ST_Geometry 类型的坐标维度。
- **ST_Dimension** - ST_Geometry 类型的维度。
- **ST_Dump** - Returns a set of geometry_dump rows for the components of a geometry.
- **ST_DumpPoints** - 3 维几何体的点集。
- **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 轴拉伸指定的距离。
- **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_ForceLHR** - LHR(Left Hand Reverse; 左手规则) 强制将几何体转换为左手规则。
- **ST_ForceRHR** - 强制将几何体转换为右手规则 (orientation) 强制将几何体转换为右手规则 (Right-Hand Rule) 强制将几何体转换为右手规则。
- **ST_ForceSFS** - 强制将几何体转换为 SFS 1.1 标准。
- **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_ForceCollection** - 强制将几何体转换为集合。
- **ST_GeomFromEWKB** - EWKB(Extended Well-Known Binary) 强制将 ST_Geometry 强制转换为 EWKB。
- **ST_GeomFromEWKT** - EWKT(Extended Well-Known Text) 强制将 ST_Geometry 强制转换为 EWKT。
- **ST_GeomFromGML** - 强制将 GML 强制转换为 PostGIS 强制转换为 PostGIS。
- **ST_GeometryN** - ST_Geometry 强制转换为 ST_GeometryN。
- **ST_GeometryType** - ST_Geometry 强制转换为 ST_GeometryType。
- **=** - 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 强制转换为 B 强制转换为 TRUE 强制转换为 TRUE。
- **~=** - A 强制转换为 B 强制转换为 TRUE 强制转换为 TRUE。
- **ST_IsClosed** - LINESTRING 强制转换为 TRUE 强制转换为 TRUE。强制转换为 TRUE (强制转换为 TRUE) 强制转换为 TRUE 强制转换为 TRUE。
- **ST_IsPlanar** - 强制转换为 ST_IsPlanar。
- **ST_IsSolid** - 强制转换为 ST_IsSolid。强制转换为 ST_IsSolid。
- **ST_MakeSolid** - 强制转换为 ST_MakeSolid。强制转换为 ST_MakeSolid。强制转换为 ST_MakeSolid, 强制转换为 ST_MakeSolid 强制转换为 TIN 强制转换为 TIN。
- **ST_MemSize** - ST_Geometry 强制转换为 ST_MemSize。
- **ST_NPoints** - 强制转换为 ST_NPoints (强制转换为 ST_NPoints) 强制转换为 ST_NPoints。
- **ST_NumGeometries** - 强制转换为 ST_NumGeometries。强制转换为 ST_NumGeometries。
- **ST_NumPatches** - 强制转换为 ST_NumPatches。强制转换为 ST_NumPatches NULL 强制转换为 ST_NumPatches。
- **ST_PatchN** - ST_Geometry 强制转换为 ST_PatchN。
- **ST_RemoveRepeatedPoints** - Returns a version of a geometry with duplicate points removed.
- **ST_Reverse** - 强制转换为 ST_Reverse。
- **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** - 计算几何体的直线骨架 (straight skeleton) 并返回。
- **ST_Summary** - 返回几何体的摘要信息。
- **ST_SwapOrdinates** - 交换几何体的前两个坐标轴。
- **ST_Tessellate** - 将几何体分解为三角形 (tessellation) 并返回 TIN 或 TIN 列表。
- **ST_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST_Volume** - 3 维几何体的体积。返回浮点值 (范围 0 到正无穷)。
- **~(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 bounding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box contains a 2D float precision bounding box (BOX2DF).
- **&&** - A 2D BOX2DF B 2D BOX2DF TRUE 表示包含。
- **&&&** - A n 维 BOX2DF B n 维 BOX2DF TRUE 表示包含。
- **@(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_LineFromM	✓ Point		✓				
ST_MakeEnvelope	✓						
ST_MakeLine	✓		✓				
ST_MakePoint	✓		✓				
ST_MakePointM	✓						
ST_MakePolygon	✓		✓				
ST_Point	✓				✓		
ST_PointZ	✓						
ST_PointM	✓						
ST_PointZM	✓						
ST_Polygon	✓		✓		✓		
ST_TileEnvelope	✓						
ST_HexagonGrid	✓						
ST_Hexagon	✓						
ST_SquareGrid	✓						
ST_Square	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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	✓		✓	✓		✓	✓
ST_NDims	✓		✓				
ST_NPoints	✓		✓	✓		✓	
ST_NRings	✓		✓	✓			
ST_NumGeometries	✓		✓		✓	✓	✓
ST_NumInteriorRings	✓				✓		
ST_NumInteriorRing	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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_CollectionHistogram	✓	genize					
ST_CurveToLine	✓		✓	✓	✓		
ST_Scroll	✓		✓				
ST_FlipCoordinates	✓		✓	✓		✓	✓
ST_Force2D	✓		✓	✓		✓	
ST_Force3D	✓		✓	✓		✓	
ST_Force3DZ	✓		✓	✓		✓	
ST_Force3DM	✓			✓			
ST_Force4D	✓		✓	✓			
ST_ForceCollection	✓		✓	✓		✓	
ST_ForceCurve	✓		✓	✓			
ST_ForcePolygonCW	✓		✓				
ST_ForcePolygonCCW	✓		✓				
ST_ForceSFS	✓		✓	✓		✓	✓
ST_ForceRHR	✓		✓			✓	
ST_LineExtend	✓						
ST_LineToCurve	✓		✓	✓			
ST_Multi	✓						
ST_Normalize	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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_SwapOrdinates	✓		✓	✓		✓	✓
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	✓						
ST_GeogFromText		✓					
ST_GeographyFromText		✓					
ST_GeomCollFromText	✓				✓		
ST_GeomFromEWKT	✓		✓	✓		✓	✓
ST_GeomFromMRC21	✓						
ST_GeometryFromText	✓				✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_GeomFromText	✓			✓	✓		
ST_LineFromText	✓				✓		
ST_MLineFromText	✓				✓		
ST_MPointFromText	✓				✓		
ST_MPolyFromText	✓				✓		
ST_PointFromText	✓				✓		
ST_PolygonFromText	✓				✓		
ST_WKTToSQL	✓				✓		
ST_GeogFromWKB		✓		✓			
ST_GeomFromEWKB	✓		✓	✓		✓	✓
ST_GeomFromWKB	✓			✓	✓		
ST_LineFromWKB	✓				✓		
ST_LinestringFromWKB	✓				✓		
ST_PointFromWKB	✓		✓	✓	✓		
ST_WKBToSQL	✓				✓		
ST_Box2dFromGeoHash		✓					
ST_GeomFromGeoHash	✓						
ST_GeomFromGeoJSON	✓		✓			✓	✓
ST_GeomFromGeoJSON	✓		✓				
ST_GeomFromKML	✓		✓				
ST_GeomFromInterpolatedPolyline	✓						
ST_PointFromGeoHash							
ST_FromFlatGeobufToTable							
ST_FromFlatGeobuf							
ST_AsEWKT	✓	✓	✓	✓		✓	✓
ST_AsText	✓	✓		✓	✓		
ST_AsBinary	✓	✓	✓	✓	✓	✓	✓
ST_AsEWKB	✓		✓	✓		✓	✓
ST_AsHEXEWKB	✓		✓	✓			
ST_AsEncodedPolyline	✓						
ST_AsFlatGeobuf							
ST_AsGeobuf							
ST_AsGeoJSON	✓	✓	✓				

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_AsGML	✓	✓	✓		✓	✓	✓
ST_AsKML	✓	✓	✓				
ST_AsLatLonText	✓						
ST_AsMARC21Text	✓						
ST_AsMVTGeoJSON	✓						
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)	✓			✓		✓	
@(box2df,box2df)	✗			✓		✓	
&>	✓						
>>	✓						
~	✓						
~(geometry,box2df)	✓			✓		✓	
~(box2df,geometry)	✓			✓		✓	

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
~(box2df,box2d)	✓			✓		✓	
~=	✓					✓	
<->	✓	✓					
=	✓						
<#>	✓						
<<->>	✓						
ST_3DIntersect	✓		✓		✓	✓	✓
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	✓	✓			✓	✓	
ST_Azimuth	✓	✓					
ST_Angle	✓						
ST_ClosestPoint	✓	✓					
ST_3DClosestPoint	✓		✓			✓	
ST_Distance	✓	✓		✓	✓		
ST_3DDistance	✓		✓		✓	✓	

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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	✓		✓				
ST_Union	✓		✓		✓		
ST_Buffer	✓	😬			✓		
ST_BuildArea	✓						
ST_Centroid	✓	✓			✓		
ST_ChaikinSmoothing	✓		✓				
ST_ConcaveHull	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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_CoverageIntersectionEdges	✓						
ST_CoverageSimplify	✓						
ST_CoverageUnion	✓						
ST_Affine	✓		✓	✓		✓	✓
ST_Rotate	✓		✓	✓		✓	✓
ST_RotateX	✓		✓			✓	✓
ST_RotateY	✓		✓			✓	✓
ST_RotateZ	✓		✓	✓		✓	✓
ST_Scale	✓		✓	✓		✓	✓
ST_Translate	✓		✓	✓			

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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	✓				✓		
ST_LocateBetween	✓				✓		
ST_LocateBetweenElevations	✓		✓				
ST_InterpolatePoint	✓		✓				
ST_AddMeasure	✓		✓				
ST_IsValidTrajectory	✓		✓				
ST_ClosestPointApproach	✓		✓				
ST_DistanceCPA	✓		✓				

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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							
ST_3DArea							
ST_Volume							
CG_Intersection							
CG_Intersects							
CG_3DIntersec							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
CG_Difference							
ST_3DDifference							
CG_3DDifference							
CG_Distance							
CG_3DDistance							
ST_3DConvexH							
CG_3DConvexH							
ST_3DIntersect							
CG_3DIntersect							
CG_Union							
ST_3DUnion							
CG_3DUnion							
ST_AlphaShape							
CG_AlphaShape							
CG_ApproxConvexPartition							
ST_ApproximateMedialAxis							
CG_ApproximateMedialAxis							
ST_ConstrainedDelaunayTriangles							
CG_ConstrainedDelaunayTriangles							
ST_Extrude							
CG_Extrude							
CG_ExtrudeStreetsSkeleton							
CG_GreeneApproxConvexPartition							
ST_MinkowskiS							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
CG_Minkowski	￼						
ST_OptimalAlp	￼ shape						
CG_OptimalAlp	￼ shape						
CG_OptimalCor	￼ Partition						
CG_StraightSke	￼ on		￼			￼	￼
ST_StraightSke	￼ n		￼			￼	￼
ST_Tessellate	￼		￼			￼	￼
CG_Tessellate	￼		￼			￼	￼
CG_Triangulate	￼						
CG_Visibility	￼		￼			￼	￼
CG_YMonotone	￼ ition						
CG_StraightSke	￼ onPartition						
CG_3DBuffer	￼						
CG_Rotate	￼						
CG_2DRotate	￼						
CG_3DRotate	￼		￼				
CG_RotateX	￼		￼				
CG_RotateY	￼		￼				
CG_RotateZ	￼		￼				
CG_Scale	￼						
CG_3DScale	￼		￼				
CG_3DScaleArc	￼ lCenter		￼				
CG_Translate	￼						
CG_3DTranslat	￼		￼				

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
CG_Simplify							
CG_3DAlphaWrapping							
getfaceedges_returntype							
TopoGeometry							
validatetopology_returntype							
TopoElement							
TopoElementArray							
AddTopoGeometryColumn							
RenameTopoGeometryColumn							
DropTopology							
RenameTopology							
DropTopoGeometryColumn							
Populate_Topology_Layer							
TopologySummary							
ValidateTopology✓							
ValidateTopologyRelation							
ValidateTopology✓recision							
MakeTopologyP✓ise							
FindTopology							
FindLayer							
TotalTopologySize							
CreateTopology							
CopyTopology							
ST_InitTopoGeo					✓		
ST_CreateTopoG✓					✓		
TopoGeo_AddPc✓							
TopoGeo_AddLi✓string							
TopoGeo_AddPc✓on							
TopoGeo_LoadC✓netry							
ST_AddIsoNode✓					✓		
ST_AddIsoEdge✓					✓		
ST_AddEdgeNe✓aces					✓		
ST_AddEdgeMo✓ice					✓		
ST_RemEdgeNewFace					✓		
ST_RemEdgeModFace					✓		
ST_ChangeEdge✓om					✓		
ST_ModEdgeSp✓					✓		
ST_ModEdgeHeal					✓		
ST_NewEdgeHeal					✓		
ST_MoveIsoNoc✓					✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_NewEdgesS	✓				✓		
ST_RemoveIsoNode					✓		
ST_RemoveIsoEdge					✓		
GetEdgeByPoint	✓						
GetFaceByPoint	✓						
GetFaceContainsPoint	✓						
GetNodeByPoint	✓						
GetTopologyID							
GetTopologySRID							
GetTopologyName							
ST_GetFaceEdges					✓		
ST_GetFaceGeometry	✓ try				✓		
GetRingEdges							
GetNodeEdges							
Polygonize							
AddNode	✓						
AddEdge	✓						
AddFace	✓						
ST_Simplify	✓						
RemoveUnusedNodes	✓						
CreateTopoGeom	✓						
toTopoGeom	✓						
TopoElementArray_Agg							
TopoElement	✓						
clearTopoGeom	✓						
TopoGeom_addLineament	✓						
TopoGeom_removeLineament	✓						
TopoGeom_addTopoGeom	✓						
toTopoGeom							
GetTopoGeomElementArray							
GetTopoGeomElements							
ST_SRID	✓				✓		
AsGML	✓						
AsTopoJSON	✓						
Equals	✓		✓				
Intersects	✓		✓				
geomval							
addbandarg							
rastbandarg							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
raster							
reclassarg							
summarystats							
unionarg							
AddRasterConstraints							
DropRasterConstraints							
AddOverviewConstraints							
DropOverviewConstraints							
PostGIS_GDAL_Version							
PostGIS_Raster_Lib_Build_Date							
PostGIS_Raster_Lib_Version							
ST_GDALDrivers							
ST_Contour							
ST_InterpolateF✓.er							
UpdateRasterSRID							
ST_CreateOverview							
ST_AddBand							
ST_AsRaster ✓							
ST_AsRasterAg✓							
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							
ST_UpperLeftX							
ST_UpperLeftY							
ST_Width							
ST_WorldToRas✓Coord							
ST_WorldToRas✓CoordX							
ST_WorldToRas✓CoordY							
ST_BandMetaData							
ST_BandNoDataValue							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_BandIsNoData							
ST_BandPath							
ST_BandFileSize							
ST_BandFileTimestamp							
ST_BandPixelType							
ST_MinPossibleValue							
ST_HasNoBand							
ST_PixelAsPolygons	✓						
ST_PixelAsPolygons							
ST_PixelAsPoint	✓						
ST_PixelAsPoints							
ST_PixelAsCentroids	✓ 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							
ST_Histogram							
ST_Quantile							
ST_SummaryStats							
ST_SummaryStatsAgg							
ST_ValueCount							
ST_RastFromWKB							
ST_RastFromHexWKB							
ST_AsBinary/ST_AsWKB							
ST_AsHexWKB							
ST_AsGDALRaster							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_AsJPEG							
ST_AsPNG							
ST_AsTIFF							
ST_Clip	✓						
ST_ColorMap							
ST_Grayscale							
ST_Intersection	✓						
ST_MapAlgebra (callback function version)							
ST_MapAlgebra (expres- sion version)							
ST_MapAlgebraExpr							
ST_MapAlgebraExpr							
ST_MapAlgebraFct							
ST_MapAlgebraFct							
ST_MapAlgebraFctNgb							
ST_Reclass							
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							
Box3D	☒						
ST_ConvexHull	✓						
ST_DumpAsPolygons							
ST_Envelope	✓						
ST_MinConvexH	✓						
ST_Polygon	✓						
&&	✓						
&<							
&>							
=							
@	✓						
~=							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
~	✓						
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_Intersection	✓						
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							
Pagc Normalize_Address							
Pprint_Addy							
Reverse_Geocode	✓						
Topology_Load_Tiger							
Set_Geocode_Setting							

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.

- ### 13.12.2 PostGIS Functions new or enhanced in 3.5

Functions new in PostGIS 3.5

- **CG_3DArea** - Availability: 3.5.0 3 ████████████████████. ████████ 0 ████████████████.
- **CG_3DConvexHull** - Availability: 3.5.0 ████████████████████████████.

- **CG_3DDifference** - Availability: 3.5.0 3D 几何体之间的差集。
- **CG_3DDistance** - Availability: 3.5.0 计算两个几何体之间的最小 3D 距离。
- **CG_3DIntersection** - Availability: 3.5.0 3D 几何体之间的交集。
- **CG_3DIntersects** - Availability: 3.5.0 测试两个 3D 几何体是否相交。
- **CG_3DUnion** - Availability: 3.5.0 使用 postgis_sfcgal 进行 3D 并集。
- **CG_AlphaShape** - Availability: 3.5.0 - requires SFCGAL >= 1.4.1. 计算一个几何体的 Alpha 形状。
- **CG_ApproxConvexPartition** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. 计算多边形几何体的近似凸分区。
- **CG_ApproximateMedialAxis** - Availability: 3.5.0 计算几何体的近似 medial axis。
- **CG_Area** - Availability: 3.5.0 计算几何体的面积。
- **CG_Difference** - Availability: 3.5.0 计算两个几何体之间的几何差集。
- **CG_Distance** - Availability: 3.5.0 计算两个几何体之间的最小距离。
- **CG_Extrude** - Availability: 3.5.0 沿指定距离拉伸几何体。
- **CG_ExtrudeStraightSkeleton** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. 直骨架拉伸。
- **CG_ForceLHR** - Availability: 3.5.0 LHR(Left Hand Reverse; 左手规则) 强制使用。
- **CG_GreeneApproxConvexPartition** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. 计算多边形几何体的近似凸分区。
- **CG_Intersection** - Availability: 3.5.0 计算两个几何体的交集。
- **CG_Intersects** - Availability: 3.5.0 测试两个几何体是否相交 (它们至少有一个公共点)。
- **CG_IsPlanar** - Availability: 3.5.0 测试几何体是否是平面。
- **CG_IsSolid** - Availability: 3.5.0 测试几何体是否是实心体。
- **CG_MakeSolid** - Availability: 3.5.0 将几何体转换为实心体。支持 TIN 格式。
- **CG_MinkowskiSum** - Availability: 3.5.0 计算两个几何体的 Minkowski 和。
- **CG_OptimalAlphaShape** - Availability: 3.5.0 - requires SFCGAL >= 1.4.1. 计算一个几何体的 Alpha 形状，使用“最优”alpha 值。
- **CG_OptimalConvexPartition** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. 计算多边形几何体的最优凸分区。
- **CG_Orientation** - Availability: 3.5.0 返回几何体的朝向 (orientation)。
- **CG_StraightSkeleton** - Availability: 3.5.0 计算几何体的直骨架 (straight skeleton)。
- **CG_Tessellate** - Availability: 3.5.0 对几何体进行三角剖分 (tessellation)。
- **CG_Triangulate** - Availability: 3.5.0 三角化一个多边形几何体。
- **CG_Union** - Availability: 3.5.0 计算两个几何体的并集。
- **CG_Visibility** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. 计算从一个点或一个段在多边形几何体中的可见性多边形。

- **CG_Volume** - Availability: 3.5.0 3D volume calculation. Returns the volume of a 3D geometry (0 to infinity).
- **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.

- ## 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, resampling_algorithm, 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. 返回最短的线。

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 返回3D凸包。
- **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.
- **TopoGeom_addTopoGeom** - Availability: 3.2 Adds element of a TopoGeometry to the definition of another TopoGeometry.
- **ValidateTopologyRelation** - Availability: 3.2.0 Returns info about invalid topology relation records
- **postgis.gdal_vsi_options** - Availability: 3.2.0 DB ￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼.

Functions enhanced in PostGIS 3.2

- **GetFaceByPoint** - Enhanced: 3.2.0 more efficient implementation and clearer contract, stops working with invalid topologies. Finds face intersecting a given point.
- **ST_ClusterKMeans** - Enhanced: 3.2.0 Support for max_radius Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST_MakeValid** - Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS >= 3.10.0. Attempts to make an invalid geometry valid without losing vertices.
- **ST_PixelAsCentroid** - ￼￼￼: 2.1.0 ￼￼￼ C ￼￼￼￼￼￼￼￼. ￼￼￼￼￼￼￼￼￼￼ (￼￼￼￼) ￼￼￼￼.
- **ST_PixelAsCentroids** - ￼￼￼: 2.1.0 ￼￼￼ C ￼￼￼￼￼￼￼￼. ￼￼￼￼￼￼￼￼￼￼ (￼￼￼￼) ￼￼￼￼￼￼ X, Y ￼￼￼￼￼￼￼￼￼. ￼￼￼￼￼￼￼￼￼￼￼￼￼￼.

- **ST_Point** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y and SRID values.
- **ST_PointM** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, M and SRID values.
- **ST_PointZ** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z and SRID values.
- **ST_PointZM** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z, M and SRID values.
- **ST_RemovePoint** - Enhanced: 3.2.0 Remove a point from a linestring.
- **ST_RemoveRepeatedPoints** - Enhanced: 3.2.0 Returns a version of a geometry with duplicate points removed.
- **ST_StartPoint** - Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString. Returns the first point of a LineString.
- **ST_Value** - `ST_Value(geometry, column_name, row_id)`: 2.1.0 `ST_Value(geometry, column_name, row_id)` exclude_nodata_value `boolean`. `column_name` `text`, `row_id` `integer`. `ST_Value(geometry, column_name, row_id)` exclude_nodata_value `boolean`, `nodata` `text`. `ST_Value(geometry, column_name, row_id)` exclude_nodata_value `boolean`, `nodata` `text`. `ST_Value(geometry, column_name, row_id)` exclude_nodata_value `boolean`, `nodata` `text`.
- **TopoGeo_AddLineString** - Enhanced: 3.2.0 added support for returning signed identifier. Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.

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_return_type 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 `ST_Hexagon(edge_size, cell_coord)`. Returns a single hexagon, using the provided edge size and cell coordinate within the hexagon grid space.
- **ST_HexagonGrid** - 2.1.0 `ST_HexagonGrid(geometry)`. Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.
- **ST_MaximumInscribedCircle** - Availability: 3.1.0. `ST_MaximumInscribedCircle(geometry)`.
- **ST_ReducePrecision** - Availability: 3.1.0. Returns a valid geometry with points rounded to a grid tolerance.
- **ST_Square** - 2.1.0 `ST_Square(edge_size, cell_coord)`. Returns a single square, using the provided edge size and cell coordinate within the square grid space.

- **ST_Count** - 2.2.0 `ST_Count(rastertable, rastercolumn, ...)` `exclude_nodata_value` 1 `exclude_nodata_value` NODATA.
- **ST_Force3D** - Changed: 3.1.0. Added support for supplying a non-zero Z value. `XYZ` `XYZ`.
- **ST_Force3DM** - Changed: 3.1.0. Added support for supplying a non-zero M value. `XYM` `XYM`.
- **ST_Force3DZ** - Changed: 3.1.0. Added support for supplying a non-zero Z value. `XYZ` `XYZ`.
- **ST_Force4D** - Changed: 3.1.0. Added support for supplying non-zero Z and M values. `XYZM` `XYZM`.
- **ST_Histogram** - Changed: 3.1.0 Removed `ST_Histogram(table_name, column_name)` variant. `(bin; ...)` `(bin; ...)`.
- **ST_Quantile** - Changed: 3.1.0 Removed `ST_Quantile(table_name, column_name)` variant. `(population) ... (quantile) ...` `(percentile) ...`.
- **ST_SummaryStats** - 2.2.0 `ST_SummaryStats(rastertable, rastercolumn, ...)` `count, sum, mean, stddev, min, max` `count, sum, mean, stddev, min, max`.

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 新增函数。Return a constrained Delaunay triangulation around the given input geometry.
- **ST_3DLineInterpolatePoint** - 2.1.0 新增函数。Returns a point interpolated along a 3D line at a fractional location.
- **ST_ConstrainedDelaunayTriangles** - 2.1.0 新增函数。Return a constrained Delaunay triangulation around the given input geometry.
- **ST_TileEnvelope** - 2.1.0 新增函数。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 转 PostGIS 函数。
- **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

- ## Functions changed in PostGIS 3.0

- ### 13.12.8 PostGIS Functions new or enhanced in 2.5

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. `ST_AsText(geometry, precision)` WKT(Well-Known Text) `SRID=SRID;POINT(x y)`.
- **ST_BandMetaData** - Enhanced: 2.5.0 to include outdbbandnum, filesize and filetimestamp for outdb rasters. `ST_BandMetaData(raster, bandnum, outdbbandnum, filesize, filetimestamp)`. `ST_BandMetaData(raster, bandnum)` 1 `ST_BandMetaData(raster)`.
- **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_GeomFromGeoJSON(json)` PostGIS `ST_GeomFromGeoJSON(jsonb)`.
- **ST_GeometricMedian** - Enhanced: 2.5.0 Added support for M as weight of points. `ST_GeometricMedian(geometry, weight)` (median) `ST_GeometricMedian(geometry)`.
- **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 `ST_AsGeobuf(geometry, options)`. Return a Geobuf representation of a set of rows.
- **ST_AsMVT** - 2.2.0 `ST_AsMVT(geometry, name, options)`. Aggregate function returning a MVT representation of a set of rows.
- **ST_AsMVTGeom** - 2.2.0 `ST_AsMVTGeom(geometry, tilebbox, options)`. Transforms a geometry into the coordinate space of a MVT tile.
- **ST_Centroid** - Availability: 2.4.0 support for geography was introduced. `ST_Centroid(geometry)`.
- **ST_ForcePolygonCCW** - 2.2.0 `ST_ForcePolygonCCW(geometry)`. Orients all exterior rings counter-clockwise and all interior rings clockwise.
- **ST_ForcePolygonCW** - 2.2.0 `ST_ForcePolygonCW(geometry)`. Orients all exterior rings clockwise and all interior rings counter-clockwise.
- **ST_FrechetDistance** - Availability: 2.4.0 - requires GEOS >= 3.7.0 `ST_FrechetDistance(geometry1, geometry2, num_points)` (shortest) `ST_FrechetDistance(geometry1, geometry2)`.
- **ST_IsPolygonCCW** - 2.2.0 `ST_IsPolygonCCW(geometry)`. Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.
- **ST_IsPolygonCW** - 2.2.0 `ST_IsPolygonCW(geometry)`. Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.
- **ST_MakeEmptyCoverage** - 2.2.0 `ST_MakeEmptyCoverage(coverage, width, height)`. 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. `zcta5_all`, `zcta5`.
- **Normalize_Address** - Enhanced: 2.4.0 norm_addy object includes additional fields zip4 and address_alphanumeric. `norm_addy`, `zip4`, `address_alphanumeric`, `norm_addy` `zip4`. `tiger_geocoder` `tiger_geocoder` (TIGER `tiger_geocoder`) `tiger_geocoder`.

- **Page_Normalize_Address** - Enhanced: 2.4.0 norm_addy object includes additional fields zip4 and address_alphanumeric. norm_addy norm_addy (TIGER geocoder (TIGER geocoder) address_standardizer).
- **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. include_strnum_range = true.
- **ST_AsTWKB** - Enhanced: 2.4.0 memory and speed improvements. TWKB(Tiny Well-Known Binary).
- **ST_Covers** - Enhanced: 2.4.0 Support for polygon in polygon and line in polygon added for geography type Tests if every point of B lies in A
- **ST_CurveToLine** - Enhanced: 2.4.0 added support for max-deviation and max-angle tolerance, and for symmetric output. Converts a geometry containing curves to a linear geometry.
- **ST_Project** - Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth. Returns a point projected from a start point by a distance and bearing (azimuth).
- **ST_Reverse** - Enhanced: 2.4.0 support for curves was introduced.

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. (robustness) (clearance).
- **ST_MinimumClearanceLine** - 2.3.0. GEOS 3.6.0. 2 , .
- **ST_Normalize** - 2.3.0. .
- **ST_Points** - 2.3.0. .
- **ST_VoronoiLines** - 2.3.0. Returns the boundaries of the Voronoi diagram of the vertices of a geometry.
- **ST_VoronoiPolygons** - 2.3.0. Returns the cells of the Voronoi diagram of the vertices of a geometry.
- **ST_WrapX** - Availability: 2.3.0 requires GEOS X .
- **TopoGeom_addElement** - 2.3. Adds an element to the definition of a TopoGeometry.
- **TopoGeom_remElement** - 2.3. Removes an element from the definition of a TopoGeometry.

- **~(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) contains another 2D float precision bounding box (BOX2DF).
- **~(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) contains a geometry's 2D bonding 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 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** - 简体中文: 2.3.0 支持负索引 (-1 到 -N) 获取几何体中的点。ST_LineString 和 ST_CircularString 均支持。

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 支持浮点精度。PostgreSQL 9.1 支持 KNN 索引。Returns the n-D distance between the A and B geometries or bounding boxes
- **ST_3DDifference** - 2.2.0 支持 3D 差集。3 个参数。
- **ST_3DUnion** - 2.2.0 支持 3D 并集。Perform 3D union.
- **ST_ApproximateMedialAxis** - 2.2.0 支持近似 medial axis。Returns the approximate medial axis of a geometry.

- **ST_AsEncodedPolyline** - 2.2.0 函数。将几何体转换为编码的多线格式。
- **ST_AsTWKB** - 2.2.0 函数。将几何体转换为 TWKB (Tiny Well-Known Binary) 格式。
- **ST_BoundingDiagonal** - 2.2.0 函数。返回几何体的边界对角线。
- **ST_CPAWithin** - 2.2.0 函数。Tests if the closest point of approach of two trajectories is within the specified distance.
- **ST_ClipByBox2D** - 2.2.0 函数。Computes the portion of a geometry falling within a rectangle.
- **ST_ClosestPointOfApproach** - 2.2.0 函数。Returns a measure at the closest point of approach of two trajectories.
- **ST_ClusterIntersecting** - 2.2.0 函数。Aggregate function that clusters input geometries into connected sets.
- **ST_ClusterWithin** - 2.2.0 函数。Aggregate function that clusters geometries by separation distance.
- **ST_CountAgg** - 2.2.0 函数。对几何体进行聚合计数。exclude_nodata_value 参数用于处理 NO DATA 值。
- **ST_CreateOverview** - 2.2.0 函数。创建几何体的概述图。
- **ST_DistanceCPA** - 2.2.0 函数。Returns the distance between the closest point of approach of two trajectories.
- **ST_ForceCurve** - 2.2.0 函数。将几何体强制转换为曲线 (upcast)。
- **ST_IsPlanar** - 2.2.0 函数。Tests if the geometry is planar. 2.1.0 版本引入了对 2.1 版本的支持。
- **ST_IsSolid** - 2.2.0 函数。Tests if the geometry is a solid.
- **ST_IsValidTrajectory** - 2.2.0 函数。Tests if the geometry is a valid trajectory.
- **ST_LineFromEncodedPolyline** - 2.2.0 函数。从编码的多线 (polyline) 格式创建几何体。
- **ST_MakeSolid** - 2.2.0 函数。将几何体转换为实体。TIN 格式的支持。
- **ST_MapAlgebra (callback function version)** - 2.2.0 函数。mask 参数用于指定掩膜。1 表示有效，0 表示无效。
- **ST_MemSize** - 2.2.0 函数。返回几何体的内存大小 (字节)。
- **ST_RemoveRepeatedPoints** - 2.2.0 函数。Returns a version of a geometry with duplicate points removed.
- **ST_Retile** - 2.2.0 函数。将几何体重新分块。
- **ST_SetEffectiveArea** - 2.2.0 函数。Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.
- **ST_SimplifyVW** - 2.2.0 函数。Returns a simplified representation of a geometry, using the Visvalingam-Whyatt algorithm.

- **ST_Subdivide** - 2.2.0 新增。Computes a rectilinear subdivision of a geometry.
- **ST_SummaryStatsAgg** - 2.2.0 新增。Aggregate function that returns count, sum, mean, stddev, min, max for a set of values. The values are aggregated by 1 dimension.
- **ST_SwapOrdinates** - 2.2.0 新增。Swaps the first and second ordinates of a geometry.
- **ST_Volume** - 2.2.0 新增。3D geometry volume. Returns the volume (in cubic units) of a 3D geometry. If the geometry is not 3D, it returns 0.
- **parse_address** - 2.2.0 新增。Parses a street address into its components.
- **postgis.enable_outdb_rasters** - 2.2.0 新增。DB connection parameter to enable out-of-database rasters.
- **postgis.gdal_datapath** - 2.2.0 新增。GDAL data path. GDAL_DATA environment variable. If not set, it defaults to the GDAL_DATA environment variable.
- **postgis.gdal_enabled_drivers** - 2.2.0 新增。PostGIS GDAL enabled drivers. GDAL_SKIP environment variable.
- **standardize_address** - 2.2.0 新增。Standardizes a street address. Returns the standardized address using the standard address format.
- **|=** - 2.2.0 新增。PostgreSQL 9.5 index-supported (index-supported) operator. A B (closest point of approach) trajectory (trajectory) operator.

Functions enhanced in PostGIS 2.2

- **<->** - 2.2.0 新增。PostgreSQL 9.5 KNN("K nearest neighbor") operator. Returns the KNN operator. PostgreSQL 9.4 KNN operator, A B 2 dimensions.
- **AsTopoJSON** - 2.2.1 新增。TopoGeometry TopoJSON operator.
- **ST_Area** - 2.2.0 新增。GeographicLib Proj 4.9.0 area operator. Returns the area of a geometry.
- **ST_AsX3D** - 2.2.0 新增。x/y, z (x/y, z) 3D XML operator. ISO-IEC-19776-1.2-X3DEncodings-XML operator.
- **ST_Azimuth** - 2.2.0 新增。GeographicLib Proj 4.9.0 azimuth operator. Returns the azimuth of a geometry.
- **ST_Distance** - 2.2.0 新增。GeographicLib Proj 4.9.0 distance operator. Returns the distance of a geometry (longest) operator.
- **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** - 2.2.0 新增。TIN (curve) operator. Returns the summary of a geometry.

- <->** - **<->**: 2.2.0 **<->** -- PostgreSQL 9.5 **<->** (Hybrid syntax) **<->** PostgreSQL 2.2 **<->**, PostgreSQL 9.5 **<->** **<->**. **<->** A **<->** B **<->** 2 **<->**.
- Get_Geocode_Setting** - **<->**: 2.2.0 **<->** geocode_settings_default **<->** **<->**. **<->** geocode_settings **<->**, **<->** geocode_settings **<->** **<->**. tiger.geocode_settings **<->**.
- ST_3DClosestPoint** - **<->**: 2.2.0 **<->** 2D **<->**, (**<->** Z **<->** 0 **<->** **<->**) 2D **<->**. 2D **<->** 3D **<->**, **<->** Z **<->** Z **<->** 0 **<->**. g2 **<->** g1 **<->** 3 **<->**. **<->** 3D **<->**.
- ST_3DDistance** - **<->**: 2.2.0 **<->**, 2D **<->** 3D **<->** Z **<->** Z **<->** 0 **<->** **<->**. **<->**, **<->** (SRS **<->**) 3 **<->**.
- ST_3DLongestLine** - **<->**: 2.2.0 **<->** 2D **<->**, (**<->** Z **<->** 0 **<->** **<->**) 2D **<->**. 2D **<->** 3D **<->**, **<->** Z **<->** Z **<->** 0 **<->**. **<->** 3 **<->** (longest) **<->**.
- ST_3DMaxDistance** - **<->**: 2.2.0 **<->**, 2D **<->** 3D **<->** Z **<->** Z **<->** 0 **<->** **<->**. **<->**, **<->** (SRS **<->**) 3 **<->**.
- ST_3DShortestLine** - **<->**: 2.2.0 **<->** 2D **<->**, (**<->** Z **<->** 0 **<->** **<->**) 2D **<->**. 2D **<->** 3D **<->**, **<->** Z **<->** Z **<->** 0 **<->**. **<->** 3 **<->** (shortest) **<->**.
- ST_DistanceSphere** - **<->**: 2.2.0 **<->** ST_Distance_Sphere **<->**. **<->** **<->**. PostgreSQL 1.5 **<->**.
- ST_DistanceSpheroid** - **<->**: 2.2.0 **<->** ST_Distance_Spheroid **<->**. **<->** **<->**. PostgreSQL 1.5 **<->**.
- 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** - **<->**: 2.2.0 **<->** ST_Length_Spheroid **<->**, ST_3DLeng **<->**. **<->**.
- ST_MemSize** - Changed: 2.2.0 name changed to ST_MemSize to follow naming convention. ST_Geome **<->**.
- 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** - **<->**: 2.2.0 **<->**'edge crosses node' **<->** id1 **<->** id2 **<->**. Returns a set of validate_topology_returntype objects detailing issues with topology.

The functions given below are PostGIS functions that were added or enhanced. Functions new in PostGIS 2.1

- **= - 2.1.0** **AsTopoJSON**. A **TopoJSON** **B** **TopoJSON** **TRUE** **TopoJSON**. **TopoJSON** **TopoJSON**.
- **AsTopoJSON** - 2.1.0 **TopoGeometry** **TopoJSON** **TopoJSON**.

- **Drop_Nation_Tables_Generate_Script** - 2.1.0 新增。生成 county_all, state_all 表，并生成 county, state 表 (州) 的 schema。
- **Get_Geocode_Setting** - 2.1.0 新增。tiger.geocode_settings 表。
- **Loader_Generate_Nation_Script** - 2.1.0 新增。生成 nation 表，并生成 nation 表的 schema。
- **Page Normalize Address** - 2.1.0 新增。生成 norm_addy 表，并生成 tiger_geocoder 表 (TIGER 表) 的 schema。
- **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 表 BOX2D 表。
- **ST_ColorMap** - 2.1.0 新增。生成 8BUI 表 (grayscale, RGB, RGBA) 表。
- **ST_Contains** - 2.1.0 新增。rastA 表 rastB 表。
- **ST_ContainsProperly** - 2.1.0 新增。rastB 表 rastA 表。
- **ST_CoveredBy** - 2.1.0 新增。rastA 表 rastB 表。
- **ST_Covers** - 2.1.0 新增。rastB 表 rastA 表。
- **ST_DFullyWithin** - 2.1.0 新增。rastA 表 rastB 表。
- **ST_DWithin** - 2.1.0 新增。rastA 表 rastB 表。
- **ST_DelaunayTriangles** - 2.1.0 新增。Returns the Delaunay triangulation of the vertices of a geometry.
- **ST_Disjoint** - 2.1.0 新增。rastA 表 rastB 表。
- **ST_DumpValues** - 2.1.0 新增。生成 2 表。
- **ST_Extrude** - 2.1.0 新增。
- **ST_ForceLHR** - 2.1.0 新增。LHR(Left Hand Reverse; 左旋) 表。
- **ST_FromGDALRaster** - 2.1.0 新增。生成 GDAL 表。
- **ST_GeomFromGeoHash** - 2.1.0 新增。GeoHash 表。
- **ST_InvDistWeight4ma** - 2.1.0 新增。
- **ST_MapAlgebra (callback function version)** - 2.1.0 新增。生成 1 表，生成 1 表。

- **ST_MapAlgebra (expression version)** - 2.1.0 新增。返回指定表达式在栅格上的结果。 - 表达式 1 和 2 是栅格，表达式 SQL 表达式 1 和 2 是 SQL 表达式。
 - **ST_MinConvexHull** - 2.1.0 新增。返回 NODATA 栅格的最小凸包。
 - **ST_MinDist4ma** - 2.1.0 新增。返回指定栅格的最小距离 (曼哈顿距离) 栅格。
 - **ST_MinkowskiSum** - 2.1.0 新增。返回 Minkowski 和。
 - **ST_NearestValue** - 2.1.0 新增。columnx 和 rowy 是栅格，返回 NODATA 栅格。
 - **ST_Neighborhood** - 2.1.0 新增。columnx 和 rowy 是栅格，返回 NODATA 栅格。
 - **ST_NotSameAlignmentReason** - 2.1.0 新增。返回栅格是否对齐，返回 0 表示对齐，1 表示不对齐。
 - **ST_Orientation** - 2.1.0 新增。返回 orientation 值。
 - **ST_Overlaps** - 2.1.0 新增。返回 rastA 和 rastB 是否重叠。
 - **ST_PixelAsCentroid** - 2.1.0 新增。返回栅格 (像素) 的质心。
 - **ST_PixelAsCentroids** - 2.1.0 新增。返回栅格 (像素) 的质心 X, Y 值。
 - **ST_PixelAsPoint** - 2.1.0 新增。返回栅格 (像素) 的点。
 - **ST_PixelAsPoints** - 2.1.0 新增。返回栅格 (像素) 的点 X, Y 值。
 - **ST_PixelOfValue** - 2.1.0 新增。返回栅格 columnx, rowy 的像素值。
 - **ST_PointFromGeoHash** - 2.1.0 新增。GeoHash 字符串转换为点。
 - **ST_RasterToWorldCoord** - 2.1.0 新增。返回栅格 X, Y(列, 行) 值。
 - **ST_Resize** - 2.1.0 新增。GDAL 1.6.1 版本。返回指定大小/分辨率的栅格。
 - **ST_Roughness** - 2.1.0 新增。DEM 栅格的粗糙度 (roughness) 值。
 - **ST_SetValues** - 2.1.0 新增。返回指定值的栅格。
 - **ST_Simplify** - 2.1.0 新增。返回指定精度 (Douglas-Peucker) 的 TopoGeometry 值。
 - **ST_StraightSkeleton** - 2.1.0 新增。返回 (straight skeleton) 值。
 - **ST_Summary** - 2.1.0 新增。返回栅格的摘要。
 - **ST_TPI** - 2.1.0 新增。返回 (Topographic Position Index) 值。
-

- **ST_TRI** - 2.1.0 新增函数。计算地形崎岖度指数 (Terrain Ruggedness Index) 值。
- **ST_Tessellate** - 2.1.0 新增函数。对多边形进行三角剖分 (tessellation) 生成 TIN。
- **ST_Tile** - 2.1.0 新增函数。将栅格数据分割成指定大小的瓦片。
- **ST_Touches** - 2.1.0 新增函数。检查两个栅格是否接触。TRUE 表示接触。
- **ST_Union** - 2.1.0 新增函数 ST_Union(rast, unionarg) 用于合并栅格。unionarg 为 1 表示合并所有带。
- **ST_Within** - 2.1.0 新增函数。检查一个栅格是否完全在另一个栅格内部。
- **ST_WorldToRasterCoord** - 2.1.0 新增函数。将世界坐标 X, Y 转换为栅格坐标。
- **Set_Geocode_Setting** - 2.1.0 新增函数。设置地理编码相关的配置。
- **UpdateRasterSRID** - 2.1.0 新增函数。更新栅格的 SRID 值。
- **clearTopoGeom** - 2.1 新增函数。清除拓扑几何体的内容。
- **postgis_sfcgal_version** - 2.1.0 新增函数。返回 SFCGAL 的版本信息。

Functions enhanced in PostGIS 2.1

- **ST_AddBand** - 增强: 2.1.0 新增 addbandarg 参数。允许在添加波段时指定波段编号。
- **ST_AddBand** - 增强: 2.1.0 新增 DB 参数。允许在添加波段时指定数据库连接。
- **ST_AsBinary/ST_AsWKB** - 增强: 2.1.0 新增 outasin 参数。返回 Well-Known Binary (WKB) 表示的栅格。
- **ST_AsGML** - 增强: 2.1.0 新增 GML 3 ID 参数。返回 GML 2 或 GML 3 格式的栅格。
- **ST_Aspect** - 增强: 2.1.0 新增 ST_MapAlgebra() 函数，支持 interpolate_nodata 参数。
- **ST_Boundary** - 增强: 2.1.0 新增函数。返回栅格的边界。
- **ST_Clip** - 增强: 2.1.0 新增 C 参数。返回根据输入几何体裁剪后的栅格。如果指定了 band number，则只处理该波段。如果 crop 未指定或为 TRUE，则输出栅格会被裁剪。如果 touched 设置为 TRUE，则接触像素也会被包含，否则只有中心在几何体内的像素才会被包含。
- **ST_DWithin** - 增强: 2.1.0 提高了地理学测试的速度。详见 Making Geography faster for details。测试两个几何体是否在给定距离内。
- **ST_DWithin** - 增强: 2.1.0 增加了对曲线几何体的支持。测试两个几何体是否在给定距离内。

- **ST_Distance** - Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details. 支持 3 参数 (longest) 版本。
- **ST_Distance** - 新增: 2.1.0 支持 3 参数 (longest) 版本。
- **ST_Distinct4ma** - 新增: 2.1.0 支持 2 参数版本。支持 4 参数版本。
- **ST_DumpPoints** - Enhanced: 2.1.0 Faster speed. Reimplemented as native-C. 支持 2 参数版本。
- **ST_HillShade** - 新增: 2.1.0 支持 ST_MapAlgebra() 函数, 支持 interpolate_nodata 参数。支持, 支持, 支持。
- **ST_MakeValid** - Enhanced: 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 2 参数版本。支持 4 参数版本。
- **ST_Mean4ma** - 新增: 2.1.0 支持 2 参数版本。支持 4 参数版本。
- **ST_Min4ma** - 新增: 2.1.0 支持 2 参数版本。支持 4 参数版本。
- **ST_PixelAsPolygons** - 新增: 2.1.0 支持 exclude_nodata_value 参数。支持 X, Y 参数。
- **ST_Polygon** - 新增: 2.1.0 支持 (支持 C 语言)。支持 NODATA 参数。
- **ST_Range4ma** - 新增: 2.1.0 支持 2 参数版本。支持 4 参数版本。
- **ST_SameAlignment** - 新增: 2.1.0 支持。支持, 支持, 支持。
- **ST_Segmentize** - 新增: 2.1.0 支持。Returns a modified geometry/geography having no segment longer than a given distance.
- **ST_SetGeoReference** - 新增: 2.1.0 支持 ST_SetGeoReference(raster, double precision, ...) 函数。支持 6 参数版本。支持 GDAL 支持 ESRI 支持 GDAL 支持。
- **ST_SetValue** - 新增: 2.1.0 支持 ST_SetValue() 函数。支持 ST_SetValues() 函数 geomval[] 参数 (wrapper) 参数。支持 columnx, rowy 参数。支持 1 参数版本, 支持 1 参数版本。
- **ST_Slope** - 新增: 2.1.0 支持 ST_MapAlgebra() 函数, 支持 units, scale, interpolate_nodata 参数。支持 (支持) 参数。支持。
- **ST_StdDev4ma** - 新增: 2.1.0 支持 2 参数版本。支持 4 参数版本。
- **ST_Sum4ma** - 新增: 2.1.0 支持 2 参数版本。支持 4 参数版本。

- **ST_Summary** - 新增: 2.1.0 新增。返回指定表 S 的摘要信息。
- **ST_Transform** - 新增: 2.1.0 新增 ST_Transform(rast, alignto) 函数。支持 NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos 插值方法。最近邻插值。
- **ST_Union** - 新增: 2.1.0 新增 ST_Union(rast) 函数 (返回 C 类型)。返回 1 个栅格。
- **ST_Union** - 新增: 2.1.0 新增 ST_Union(rast) 函数 1 个栅格。PostGIS 返回 1 个栅格。
- **ST_Union** - 新增: 2.1.0 新增 ST_Union(rast, uniontype) 函数 4 个栅格。返回 1 个栅格。
- **toTopoGeom** - 新增: 2.1.0 新增 TopoGeometry 类型。Converts a simple Geometry into a topo geometry.

Functions changed in PostGIS 2.1

- **ST_Aspect** - 新增: 2.1.0 新增 ST_Aspect 函数。2.1.0 新增 ST_Aspect 函数。
- **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 新增, 2.0.x 新增 ST_Force_2D 函数。返回 "2 维" 数据。
- **ST_Force3D** - 新增: 2.1.0 新增, 2.0.x 新增 ST_Force_3D 函数。返回 XYZ 数据。ST_Force3DZ 函数。
- **ST_Force3DM** - 新增: 2.1.0 新增, 2.0.x 新增 ST_Force_3DM 函数。返回 XYM 数据。
- **ST_Force3DZ** - 新增: 2.1.0 新增, 2.0.x 新增 ST_Force_3DZ 函数。返回 XYZ 数据。
- **ST_Force4D** - 新增: 2.1.0 新增, 2.0.x 新增 ST_Force_4D 函数。返回 XYZM 数据。
- **ST_ForceCollection** - 新增: 2.1.0 新增, 2.0.x 新增 ST_Force_Collection 函数。
- **ST_HillShade** - 新增: 2.1.0 新增 ST_HillShade 函数。2.1.0 新增 ST_HillShade 函数。
- **ST_LineInterpolatePoint** - 新增: 2.1.0 新增, 2.0.x 新增 ST_Line_Interpolate_Point 函数。Returns a point interpolated along a line at a fractional location.
- **ST_LineLocatePoint** - 新增: 2.1.0 新增, 2.0.x 新增 ST_Line_Locate_Point 函数。Returns the fractional location of the closest point on a line to a point.
- **ST_LineSubstring** - 新增: 2.1.0 新增, 2.0.x 新增 ST_Line_Substring 函数。Returns the part of a line between two fractional locations.
- **ST_PixelAsCentroids** - 新增: 2.1.1 新增 exclude nodata value 选项。返回 X, Y 坐标。

- **ST_PixelAsPoints** - 新增: 2.1.1 新增 `exclude_nodata_value` 参数。返回栅格像素的 X, Y 坐标。如果 `exclude_nodata_value` 为 `TRUE`, 则返回的坐标将排除 `NODATA` 值。
- **ST_PixelAsPolygons** - 新增: 2.1.1 新增 `exclude_nodata_value` 参数。返回栅格像素的多边形。如果 `exclude_nodata_value` 为 `TRUE`, 则返回的多边形将排除 `NODATA` 值。
- **ST_Polygon** - 新增: 2.1.0 新增 `exclude_nodata_value` 参数。返回栅格像素的多边形。如果 `exclude_nodata_value` 为 `TRUE`, 则返回的多边形将排除 `NODATA` 值。
- **ST_RasterToWorldCoordX** - 新增: 2.1.0 新增 `ST_Raster2WorldCoordX` 函数。返回栅格像素的 X 坐标。如果 `exclude_nodata_value` 为 `TRUE`, 则返回的坐标将排除 `NODATA` 值。
- **ST_RasterToWorldCoordY** - 新增: 2.1.0 新增 `ST_Raster2WorldCoordY` 函数。返回栅格像素的 Y 坐标。如果 `exclude_nodata_value` 为 `TRUE`, 则返回的坐标将排除 `NODATA` 值。
- **ST_Rescale** - 新增: 2.1.0 新增 `SRID` 参数。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** - 新增: 2.1.0 新增 `SRID` 参数。Resample a raster by adjusting only its skew (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.
- **ST_Segmentize** - Changed: 2.1.0 As a result of the introduction of geography support, the usage `ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5)` causes an ambiguous function error. The input needs to be properly typed as a geometry or geography. Use `ST_GeomFromText`, `ST_GeogFromText` or a cast to the required type (e.g. `ST_Segmentize('LINESTRING(1 2, 3 4)::geometry, 0.5)`) Returns a modified geometry/geography having no segment longer than a given distance.
- **ST_Slope** - 新增: 2.1.0 新增 `SRID` 参数。返回栅格像素的坡度。如果 `exclude_nodata_value` 为 `TRUE`, 则返回的坡度将排除 `NODATA` 值。
- **ST_SnapToGrid** - 新增: 2.1.0 新增 `SRID` 参数。返回栅格像素的坐标。如果 `exclude_nodata_value` 为 `TRUE`, 则返回的坐标将排除 `NODATA` 值。
- **ST_WorldToRasterCoordX** - 新增: 2.1.0 新增 `ST_World2RasterCoordX` 函数。返回栅格像素的 X 坐标。如果 `exclude_nodata_value` 为 `TRUE`, 则返回的坐标将排除 `NODATA` 值。
- **ST_WorldToRasterCoordY** - 新增: 2.1.0 新增 `ST_World2RasterCoordY` 函数。返回栅格像素的 Y 坐标。如果 `exclude_nodata_value` 为 `TRUE`, 则返回的坐标将排除 `NODATA` 值。

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 新增。A 与 B 的按位与操作。如果 A 和 B 都为 `TRUE`, 则返回 `TRUE`。
- **&&&** - 2.0.0 新增。A 与 B 的按位与操作。如果 A 和 B 都为 `TRUE`, 则返回 `TRUE`。
- **<#>** - 2.0.0 新增。PostgreSQL 9.1 新增。KNN 操作。A 与 B 的按位与操作。如果 A 和 B 都为 `TRUE`, 则返回 `TRUE`。

- **<->** - 2.0.0 新增。KNN 查询。PostgreSQL 9.1 支持。A 与 B 距离 2 以内。
- **@** - 2.0.0 新增 raster @ raster, raster @ geometry 函数。B 与 A 距离 TRUE 返回。支持。
- **@** - 2.0.5 新增 geometry @ raster 函数。B 与 A 距离 TRUE 返回。支持。
- **AddEdge** - 2.0.0 新增。添加边 (edge primitive) 到面 (face) 中。返回 ID(edgeid)。
- **AddFace** - 2.0.0 新增。添加面 (face primitive) 到体 (volume) 中。返回 ID(faceid)。
- **AddNode** - 2.0.0 新增。添加节点 (node primitive) 到边 (edge) 中。返回 ID(nodeid)。
- **AddOverviewConstraints** - 2.0.0 新增。添加概视图约束 (overview)。
- **AddRasterConstraints** - 2.0.0 新增。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 新增。TopoGeometry 转 GML 格式。
- **CopyTopology** - 2.0.0 新增。Makes a copy of a topology (nodes, edges, faces, layers and TopoGeometries) into a new schema
- **DropOverviewConstraints** - 2.0.0 新增。删除概视图约束 (overview)。
- **DropRasterConstraints** - 2.0.0 新增。删除 PostGIS 栅格约束。
- **Drop_Indexes_Generate_Script** - 2.0.0 新增。TIGER 索引生成脚本。返回 tiger_data 表。
- **Drop_State_Tables_Generate_Script** - 2.0.0 新增。删除州 (州) 表。返回 tiger_data 表。
- **Geocode_Intersection** - 2.0.0 新增。返回 2 个点, 点, 返回 NAD83 坐标, geomout, 返回 normalized address (addy) 函数。返回 10 以内。TIGER 索引 (edge, face, addr) 与 PostgreSQL 索引 (soundex, levenshtein) 函数。
- **GetEdgeByPoint** - 2.0.0 新增。Finds the edge-id of an edge that intersects a given point.
- **GetFaceByPoint** - 2.0.0 新增。Finds face intersecting a given point.
- **GetNodeByPoint** - 2.0.0 新增。Finds the node-id of a node at a point location.
- **GetNodeEdges** - 2.0 新增。返回节点边。

- **GetRingEdges** - 2.0.0 返回给定拓扑几何体的所有环的边。返回的边是拓扑几何体的子集，其拓扑学元素类型为边。
- **GetTopoGeomElements** - 2.0.0 返回给定 TopoGeometry (primitive elements) 的拓扑学元素。返回的边是拓扑几何体的子集，其拓扑学元素类型为边。
- **GetTopologySRID** - 2.0.0 返回给定拓扑几何体的 topology.topology SRID。
- **Get_Tract** - 2.0.0 返回给定拓扑几何体的所有 tract (tract) 的 field (field)。
- **Install_Missing_Indexes** - 2.0.0 安装缺失的索引 (join) (key)。
- **Loader_Generate_Census_Script** - 2.0.0 生成 Census 脚本。生成脚本 (州) 的 tiger_data 文件。生成脚本 (州) 的 tiger_data 文件。生成脚本 (州) 的 tiger_data 文件。
- **Loader_Generate_Script** - 2.0.0 生成 Tiger 2010 脚本。生成脚本 (州) 的 tiger_data 文件。生成脚本 (州) 的 tiger_data 文件。生成脚本 (州) 的 tiger_data 文件。
- **Missing_Indexes_Generate_Script** - 2.0.0 生成缺失的索引脚本 (join) (key) 的 SQL DDL 脚本。
- **Polygonize** - 2.0.0 找到并注册所有由拓扑边定义的面。
- **Reverse_Geocode** - 2.0.0 反向地理编码。返回给定地址的地理坐标。include_strnum_range = true 选项，返回包含街道编号的结果。
- **ST_3DClosestPoint** - 2.0.0 返回 g2 到 g1 的 3D 最近点。
- **ST_3DDFullyWithin** - 2.0.0 测试两个 3D 几何体是否完全在给定 3D 距离内。
- **ST_3DDWithin** - 2.0.0 测试两个 3D 几何体是否在给定 3D 距离内。
- **ST_3DDistance** - 2.0.0 返回两个 3D 几何体 (SRS 不同) 的 3D 距离。
- **ST_3DIntersects** - 2.0.0 测试两个几何体在 3D 空间是否相交 (仅适用于点、线串、多边形、多面体表面 (area))。
- **ST_3DLongestLine** - 2.0.0 返回两个 3D 几何体 (longest) 的最长线。
- **ST_3DMaxDistance** - 2.0.0 返回两个 3D 几何体 (SRS 不同) 的 3D 最大距离。
- **ST_3DShortestLine** - 2.0.0 返回两个 3D 几何体 (shortest) 的最短线。
- **ST_AddEdgeModFace** - 2.0 添加边并修改面。添加边并修改面。添加边并修改面。

- **ST_AddEdgeNewFaces** - 2.0 函数。添加新的面，并返回新的面 ID。返回的面 ID 是 2 个值的数组。
- **ST_AsGDALRaster** - 2.0.0 函数。GDAL 1.6.0 函数。返回的栅格 tile 在指定的 GDAL Raster 格式中。栅格格式是那些由您的编译库支持的。使用 ST_GDALDrivers() 来获取由您的库支持的格式列表。
- **ST_AsJPEG** - 2.0.0 函数。GDAL 1.6.0 函数。将栅格数据转换为 JPEG(Joint Photographic Experts Group) 格式 (栅格格式)。返回的栅格数据是 1 个值，3 个值或 3 个值的数组。返回 3 个值的数组 3 个值的 RGB 栅格数据。
- **ST_AsLatLonText** - 2.0 函数。将栅格数据转换为文本格式，返回栅格数据。
- **ST_AsPNG** - 2.0.0 函数。GDAL 1.6.0 函数。将栅格数据转换为 PNG(Portable Network Graphics) 格式 (栅格格式)。返回的栅格数据是 1 个值，3 个值，4 个值的数组或 4 个值的数组。返回 2 个值 4 个值的数组。返回 1 个值的 RGB 或 RGBA 栅格数据。
- **ST_AsRaster** - 2.0.0 函数。GDAL 1.6.0 函数。PostGIS 栅格 PostGIS 栅格数据。
- **ST_AsTIFF** - 2.0.0 函数。GDAL 1.6.0 函数。返回的栅格数据作为单个 TIFF 图像 (字节数组)。如果没有指定带或指定的带不存在于栅格中，则将尝试使用所有带。
- **ST_AsX3D** - 2.0.0 函数。ISO-IEC-19776-1.2-X3DEncodings-XML 函数。返回 X3D XML 数据: ISO-IEC-19776-1.2-X3DEncodings-XML 数据。
- **ST_Aspect** - 2.0.0 函数。返回栅格数据的坡度 (栅格格式)。返回的栅格数据是栅格数据。
- **ST_Band** - 2.0.0 函数。返回栅格数据的带。返回的栅格数据是栅格数据。
- **ST_BandIsNoData** - 2.0.0 函数。返回 NODATA 栅格数据。
- **ST_Clip** - 2.0.0 函数。返回由输入几何裁剪的栅格。如果带号未指定，则处理所有带。如果 crop 未指定或 TRUE，则输出栅格被裁剪。如果 touched 设置为 TRUE，则包含被触及的像素，否则仅当像素的中心在几何中时才包含。
- **ST_CollectionHomogenize** - 2.0.0 函数。返回几何集合的最简单表示。
- **ST_ConcaveHull** - 2.0.0 函数。计算一个可能的凹几何，该几何包含所有输入几何顶点。
- **ST_Count** - 2.0.0 函数。返回栅格数据中的像素数。返回的栅格数据是 1 个值。exclude_nodata_value 栅格数据，NODATA 栅格数据。
- **ST_CreateTopoGeo** - 2.0 函数。将栅格数据转换为拓扑几何。
- **ST_Distinct4ma** - 2.0.0 函数。返回栅格数据的 4 个值的数组。
- **ST_FlipCoordinates** - 2.0.0 函数。返回几何的 X 和 Y 轴翻转版本。

- **ST_GDALDrivers** - 2.0.0 返回 GDAL 1.6.0 支持的栅格格式列表。只有那些 can_write=True 的格式可以被 ST_AsGDALRaster 使用。
- **ST_GeomFromGeoJSON** - 2.0.0 从 GeoJSON 0.9 格式创建几何体。
- **ST_GetFaceEdges** - 2.0 返回面 (face) 的边 (edges)。
- **ST_HasNoBand** - 2.0.0 检查栅格是否有带 (band)。如果没有带，返回 1。
- **ST_HillShade** - 2.0.0 计算地形阴影。需要高程数据。
- **ST_Histogram** - 2.0.0 计算几何体的直方图 (bin; 范围)。返回统计信息。
- **ST_InterpolatePoint** - 2.0.0 在几何体上插值点。需要 M 值 (高程)。
- **ST_IsEmpty** - 2.0.0 检查几何体是否为空 (width = 0, height = 0)。
- **ST_IsValidDetail** - 2.0.0 返回 valid_detail 行，说明几何体是否有效或无效的原因和位置。
- **ST_IsValidReason** - Availability: 2.0 version taking flags. 返回文本说明几何体是否有效或无效的原因。
- **ST_MakeLine** - 2.0.0 从点集创建线。
- **ST_MakeValid** - 2.0.0 尝试使无效的几何体有效而不丢失顶点。
- **ST_MapAlgebraExpr** - 2.0.0 使用 1 个表达式：PostgreSQL 函数、聚合函数、或 1 个输入栅格。
- **ST_MapAlgebraExpr** - 2.0.0 使用 2 个表达式：PostgreSQL 函数、聚合函数、或 1 个输入栅格。extenttype 可以是 INTERSECTION, UNION, FIRST, SECOND。
- **ST_MapAlgebraFct** - 2.0.0 使用 1 个函数：PostgreSQL 函数、聚合函数、或 1 个输入栅格。
- **ST_MapAlgebraFct** - 2.0.0 使用 2 个函数：PostgreSQL 函数、聚合函数、或 1 个输入栅格。extenttype 可以是 INTERSECTION。
- **ST_MapAlgebraFctNgb** - 2.0.0 使用 1 个函数：PostgreSQL 函数 (Map Algebra Nearest Neighbor) 或 neighborhood 函数。
- **ST_Max4ma** - 2.0.0 计算 4 邻域最大值。

- **ST_Mean4ma** - 2.0.0 返回多边形中所有点的平均值。返回一个包含所有点平均值的点。
- **ST_Min4ma** - 2.0.0 返回多边形中所有点的最小值。返回一个包含所有点最小值的点。
- **ST_ModEdgeHeal** - 2.0 修复两个边，通过删除连接它们的节点，修改第一个边并删除第二个边。返回删除的节点 ID。
- **ST_MoveIsoNode** - 2.0.0 移动拓扑中的一个孤立节点到另一个点。如果新点几何存在，则抛出错误。返回移动的描述。
- **ST_NewEdgeHeal** - 2.0 修复两个边，通过删除连接它们的节点，并用与第一个边方向相同的边替换它们。
- **ST_Node** - 2.0.0 将节点集合转换为线。
- **ST_NumPatches** - 2.0.0 返回多边形中所有面的数量。返回一个包含所有面数量的点。如果为 NULL，则返回 NULL。
- **ST_OffsetCurve** - 2.0 返回在给定距离和侧向偏移距离处从输入线偏移的线。
- **ST_PatchN** - 2.0.0 返回 ST_Geometry 中的第 N 个面。
- **ST_Perimeter** - 2.0.0 返回多边形或地理多边形的周长。
- **ST_PixelAsPolygon** - 2.0.0 返回像素的多边形。返回一个包含所有像素的多边形的点。
- **ST_PixelAsPolygons** - 2.0.0 返回像素的多边形。返回一个包含所有像素的多边形的点。X, Y 坐标。
- **ST_Project** - 2.0.0 返回从起始点按距离和方位角投影的点。
- **ST_Quantile** - 2.0.0 返回多边形中所有点的分位数。返回一个包含所有点分位数的点。分位数，例如 25%, 50%, 75% (百分位)。
- **ST_Range4ma** - 2.0.0 返回多边形中所有点的范围。返回一个包含所有点范围的点。
- **ST_Reclass** - 2.0.0 重新分类多边形。返回一个包含所有重新分类后的多边形的点。nband 值。nband 值 1 到 255。返回一个包含所有重新分类后的多边形的点。nband 值: 16BUI 到 8BUI。
- **ST_RelateMatch** - 2.0.0 测试 DE-9IM 交集矩阵是否匹配交集矩阵模式。
- **ST_RemEdgeModFace** - 2.0 移除边，如果边分隔两个面，则删除一个面并修改另一个面以覆盖两个面的空间。
- **ST_RemEdgeNewFace** - 2.0 移除边，并创建新的面。返回一个包含所有重新创建后的多边形的点。
- **ST_Resample** - 2.0.0 重新采样多边形。GDAL 1.6.1 版本。返回一个包含所有重新采样后的多边形的点。

- **ST_Rescale** - 2.0.0 新增函数。GDAL 1.6.1 版本。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 新增函数。GDAL 1.6.1 版本。对 (x, y) 坐标进行 NearestNeighbor, Bilinear, Cubic, CubicSpline 或 Lanczos 重采样。默认是 NearestNeighbor。
- **ST_SameAlignment** - 2.0.0 新增函数。检查两个栅格是否具有相同的对齐方式 (即是否都是上行对齐或都是下行对齐)。返回 true 表示对齐，false 表示不对齐。
- **ST_SetBandIsNoData** - 2.0.0 新增函数。设置 isnodata 标志。
- **ST_SharedPaths** - 2.0.0 新增函数。返回两个多线集合中共享的路径。
- **ST_Slope** - 2.0.0 新增函数。计算坡度 (度)。返回坡度值。
- **ST_Snap** - 2.0.0 新增函数。将几何体 snap 到最近的栅格像素中心。
- **ST_SnapToGrid** - 2.0.0 新增函数。GDAL 1.6.1 版本。将几何体 snap 到最近的栅格像素中心。NearestNeighbor, Bilinear, Cubic, CubicSpline 或 Lanczos 重采样。默认是 NearestNeighbor。
- **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 新增函数。计算 4 移动平均标准差。
- **ST_Sum4ma** - 2.0.0 新增函数。计算 4 移动平均和。
- **ST_SummaryStats** - 2.0.0 新增函数。计算统计摘要 (count, sum, mean, stddev, min, max)。返回 1 个记录。
- **ST_Transform** - 2.0.0 新增函数。GDAL 1.6.1 版本。将几何体从一种坐标系转换到另一种坐标系。NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos 重采样。默认是 NearestNeighbor。
- **ST_UnaryUnion** - 2.0.0 新增函数。Computes the union of the components of a single geometry.
- **ST_Union** - 2.0.0 新增函数。返回两个几何体的并集。
- **ST_ValueCount** - 2.0.0 新增函数。返回每个值及其出现次数 (按值排序)。返回 1 个记录。NODATA 值返回 0。返回 1 个记录。
- **TopoElementArray_Agg** - 2.0.0 新增函数。Returns a topoelementarray for a set of element_id, type arrays (topoelements).
- **TopoGeo_AddLineString** - 2.0.0 新增函数。Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.

- **TopoGeo_AddPoint** - 2.0.0 新增。向拓扑中添加点并分割 (split) 拓扑。返回面标识符。
- **TopoGeo_AddPolygon** - 2.0.0 新增。向现有拓扑添加多边形，使用容差并可能分割现有边/面。返回面标识符。
- **TopologySummary** - 2.0.0 新增。根据拓扑名称提供对象类型的汇总。
- **Topology_Load_Tiger** - 2.0.0 新增。PostGIS 加载 TIGER 数据。
- **toTopoGeom** - 2.0 新增。将简单几何体转换为拓扑几何体。
- **~** - 2.0.0 新增。A 与 B 不相交返回 TRUE。
- **~=** - 2.0.0 新增。A 与 B 几乎不相交返回 TRUE。

Functions enhanced in PostGIS 2.0

- **&&** - 2.0.0 新增。多面体表面 (polyhedral surface) 相交。A 与 B 相交返回 TRUE。
- **AddGeometryColumn** - 2.0.0 新增。使用 typmod 添加几何列。
- **Box2D** - 2.0.0 新增。返回几何体的 2D 包围盒。
- **Box3D** - 2.0.0 新增。返回几何体的 3D 包围盒。
- **CreateTopology** - 增强：2.0 增加了接受 hasZ 的签名，用于创建新的拓扑模式并注册到 topology.topology 表中。
- **Geocode** - 2.0.0 新增。使用 TIGER 2010 数据，根据地址返回地理坐标。支持 max_results 参数。默认使用 NAD83 坐标系。支持 restrict_region 参数。
- **GeometryType** - 2.0.0 新增。返回几何体的类型。
- **Populate_Geometry_Columns** - 2.0.0 新增。确保几何列具有适当的类型修饰符或空间约束。
- **ST_3DExtent** - 2.0.0 新增。返回几何体的 3D 包围盒。
- **ST_Affine** - 2.0.0 新增。应用 3D 仿射变换到几何体。
- **ST_Area** - 2.0.0 新增。返回多面体表面的面积。
- **ST_AsBinary** - 2.0.0 新增。返回 OGC/ISO Well-Known Binary (WKB) 表示的几何/地理数据，不含 SRID 元数据。

- [illegible]

- **ST_GeomFromGML** - 新增: 2.0.0 支持多面体表面 (polyhedral surface) 的 TIN 格式。从 GML 格式导入 PostGIS 数据库。
- **ST_GeomFromGML** - 新增: 2.0.0 支持多面体表面 SRID 的 GML 格式导入 PostGIS 数据库。
- **ST_GeometryN** - 新增: 2.0.0 支持多面体表面, 返回 TIN 格式的多面体表面。ST_Geometry 的变体。
- **ST_GeometryType** - 新增: 2.0.0 支持多面体表面 (polyhedral surface) 的格式。ST_Geometry 的变体。
- **ST_IsClosed** - 新增: 2.0.0 支持多面体表面 (polyhedral surface) 的格式。LINESTRING 格式的多面体表面返回 TRUE。多面体表面 (多面体表面) 返回 TRUE 的格式。
- **ST_MakeEnvelope** - 新增: 2.0 支持 SRID 的多面体表面 (envelope) 的格式。多面体表面。多面体表面 (多面体表面) 的格式。SRID 的格式 SRS 的格式。
- **ST_MakeValid** - Enhanced: 2.0.1, speed improvements Attempts to make an invalid geometry valid without losing vertices.
- **ST_NPoints** - 新增: 2.0.0 支持多面体表面 (polyhedral surface) 的格式。多面体表面 (多面体) 的格式。
- **ST_NumGeometries** - 新增: 2.0.0 支持多面体表面, 返回 TIN 格式的多面体表面。多面体表面 (多面体表面) 的格式。
- **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** - 新增: 2.0.0 支持多面体表面, 返回 TIN 格式的多面体表面。Rotates a geometry about an origin point.
- **ST_Rotate** - Enhanced: 2.0.0 additional parameters for specifying the origin of rotation were added. Rotates a geometry about an origin point.
- **ST_RotateX** - 新增: 2.0.0 支持多面体表面, 返回 TIN 格式的多面体表面。Rotates a geometry about the X axis.
- **ST_RotateY** - 新增: 2.0.0 支持多面体表面, 返回 TIN 格式的多面体表面。Rotates a geometry about the Y axis.
- **ST_RotateZ** - 新增: 2.0.0 支持多面体表面, 返回 TIN 格式的多面体表面。Rotates a geometry about the Z axis.
- **ST_Scale** - 新增: 2.0.0 支持多面体表面, 返回 TIN 格式的多面体表面。Scales a geometry by given factors.
- **ST_ShiftLongitude** - 新增: 2.0.0 支持多面体表面 (polyhedral surface) 的 TIN 格式。Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- **ST_Summary** - 新增: 2.0.0 支持多面体表面 (polyhedral surface) 的格式。多面体表面 (多面体表面) 的格式。
- **ST_Transform** - 新增: 2.0.0 支持多面体表面 (polyhedral surface) 的格式。Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST_Value** - 新增: 2.0.0 支持多面体表面 exclude_nodata_value 的格式。columnx, rowy 的格式, 多面体表面 (多面体表面) 的格式。返回 1 的格式, 多面体表面 (多面体表面) 的格式。exclude_nodata_value 的格式, nodata 的格式。exclude_nodata_value 的格式, 多面体表面 (多面体表面) 的格式。

- **ValidateTopology** - 新增: 2.0.0 新增验证拓扑函数，返回验证拓扑返回类型对象，返回验证拓扑返回类型对象 (false positive) 验证拓扑返回类型。Returns a set of validate_topology_return_type objects detailing issues with topology.

Functions changed in PostGIS 2.0

- **AddGeometryColumn** - 新增: 2.0.0 新增 geometry_columns 表，新增 geometry_columns 表。新增 geometry_columns 表，新增 geometry_columns 表。PostgreSQL 新增 WGS84 POINT 数据类型。ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326);
- **AddGeometryColumn** - 新增: 2.0.0 新增。新增 use_typedmod 表。
- **AddGeometryColumn** - 新增: 2.0.0 新增。新增 geometry_columns 表，新增 typedmod 表，新增 typedmod 表。新增 geometry_columns 表，新增 typedmod 表。
- **Box3D** - 新增: 2.0.0 新增 BOX3D 表 BOX2D 表。BOX2D 表，2.0.0 新增 BOX3D 表。
- **DropGeometryColumn** - 新增: 2.0.0 新增。新增 geometry_columns 表，新增 ALTER TABLE 表。
- **DropGeometryTable** - 新增: 2.0.0 新增。新增 geometry_columns 表，新增 DROP TABLE 表。
- **Populate_Geometry_Columns** - 新增: 2.0.0 新增。新增 use_typedmod 表，新增 geometry_columns 表。Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- **ST_3DExtent** - 新增: 2.0.0 新增。In prior versions this used to be called ST_Extent3D Aggregate function that returns the 3D bounding box of geometries.
- **ST_3DLength** - 新增: 2.0.0 新增 ST_Length3D 表。
- **ST_3DMakeBox** - 新增: 2.0.0 新增。In prior versions this used to be called ST_MakeBox3D Creates a BOX3D defined by two 3D point geometries.
- **ST_3DPerimeter** - 新增: 2.0.0 新增 ST_Perimeter3D 表。
- **ST_AsBinary** - 新增: 2.0.0 新增 ST_AsBinary('POINT(1 2)') 表，n st_asbinary(unknown) is not unique error 表。ST_AsBinary('POINT(1 2)::geometry'); 表。legacy.sql 表。Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST_AsGML** - 新增: 2.0.0 新增 (named arg) 表。GML 2 表 GML 3 表。
- **ST_AsGeoJSON** - 新增: 2.0.0 新增 (default arg) 表 (named arg) 表。Return a geometry or feature in GeoJSON format.
- **ST_AsSVG** - 新增: 2.0.0 新增 (default arg) 表 (named arg) 表。Returns SVG path data for a geometry.

- **ST_EndPoint** - 函数: 2.0.0 添加。PostGIS 2.0.0 之前，ST_EndPoint 返回 NULL。2.0.0 添加，ST_LineString 和 ST_CircularString 返回最后一个点。
- **ST_GDALDrivers** - 函数: 2.0.6, 2.1.3 添加 - GUC 参数 `gdal_enabled_drivers` 控制。Returns a list of raster formats supported by PostGIS through GDAL. Only those formats with `can_write=True` can be used by `ST_AsGDALRaster`
- **ST_GeomFromText** - 函数: PostGIS 2.0.0 添加。ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') 返回空几何。WKT 格式。PostGIS 2.0.0 之前，SQL/MM 返回 NULL。ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') 返回空几何。
- **ST_GeometryN** - 函数: 2.0.0 添加。ST_GeometryN(...,1) 返回第一个子几何。ST_Geometry 返回整个几何。
- **ST_IsEmpty** - 函数: PostGIS 2.0.0 添加。ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') 返回空几何。PostGIS 2.0.0 之前，SQL/MM 返回 NULL。Tests if a geometry is empty.
- **ST_Length** - 函数: 2.0.0 添加。2.0.0 之前，ST_Length 返回 NULL。2.0.0 添加，ST_Perimeter 返回周长。ST_Length 返回长度。
- **ST_LocateAlong** - 函数: 2.0.0 添加。ST_Locate Along_Measure 返回点。Returns the point(s) on a geometry that match a measure value.
- **ST_LocateBetween** - 函数: 2.0.0 添加。ST_Locate Along_Measure 返回范围。Returns the portions of a geometry that match a measure range.
- **ST_ModEdgeSplit** - 函数: 2.0 添加。ST_ModEdgesSplit 返回修改后的几何。
- **ST_NumGeometries** - 函数: 2.0.0 添加。ST_NumGeometries 返回几何数量。2.0.0 之前，返回 1。
- **ST_NumInteriorRings** - 函数: 2.0.0 添加。ST_NumInteriorRings 返回内部环数量。
- **ST_PointN** - 函数: 2.0.0 添加。PostGIS 2.0.0 之前，ST_PointN 返回 NULL。ST_LineString 和 ST_CircularString 返回第 N 个点。
- **ST_ScaleX** - 函数: 2.0.0 添加。WKTRaster 返回 ST_PixelSizeX。
- **ST_ScaleY** - 函数: 2.0.0 添加。WKTRaster 返回 ST_PixelSizeY。
- **ST_SetScale** - 函数: 2.0.0 添加。WKTRaster 返回 ST_SetPixelSize。2.0.0 之前，返回 NULL。
- **ST_StartPoint** - 函数: 2.0.0 添加。PostGIS 2.0.0 之前，ST_StartPoint 返回 NULL。2.0.0 添加，ST_LineString 返回第一个点。

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 新增函数。A 2D 几何体 B 2D 几何体 TRUE 或 FALSE。
- **PostGIS_LibXML_Version** - 1.5 新增函数。Returns the version number of the libxml2 library.
- **ST_AddMeasure** - 1.5.0 新增函数。Interpolates measures along a linear geometry.
- **ST_AsBinary** - 1.5.0 新增函数。Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST_AsGML** - 1.5.0 新增函数。返回 GML 2 或 GML 3 格式。
- **ST_AsGeoJSON** - 1.5.0 新增函数。Return a geometry or feature in GeoJSON format.
- **ST_AsText** - 1.5.0 新增函数。返回/输入 WKT(Well-Known Text) 格式 SRID 几何体。
- **ST_Buffer** - Availability: 1.5 - ST_Buffer was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added. Computes a geometry covering all points within a given distance from a geometry.
- **ST_ClosestPoint** - 1.5.0 新增函数。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 新增函数。Given a geometry collection, returns a multi-geometry containing only elements of a specified type.
- **ST_Covers** - 1.5.0 新增函数。Tests if every point of B lies in A
- **ST_DFullyWithin** - 1.5.0 新增函数。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 新增函数。返回两个几何体之间的最短距离。支持 3 种模式 (longest) 模式。
- **ST_DistanceSphere** - 1.5 新增函数。Returns the distance between two points on a sphere. PostGIS 1.5 新增函数。
- **ST_DistanceSpheroid** - 1.5 新增函数。Returns the distance between two points on a spheroid. PostGIS 1.5 新增函数。
- **ST_DumpPoints** - 1.5.0 新增函数。Returns the points of a geometry.
- **ST_Envelope** - 1.5.0 新增函数, float4 精度, float8 精度 (double precision; float8) 精度。
- **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 新增函数。LibXML2 1.6 新增函数。GML 几何体 ST_Geometry 函数。返回 ST_GeomFromGML 函数。

- **ST_GeomFromGML** - 1.5 支持 GML 2.0 和 3.0 版本。LibXML2 1.6 支持 GML 2.0 和 3.0 版本。PostGIS 1.5.0 支持 GML 2.0 和 3.0 版本。
- **ST_GeomFromKML** - Availability: 1.5, requires libxml2 2.6+ 支持 KML 2.0 和 3.0 版本。PostGIS 1.5.0 支持 KML 2.0 和 3.0 版本。
- **ST_HausdorffDistance** - 1.5.0 支持。返回两个几何体之间的 Hausdorff 距离 (shortest)。
- **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 支持。返回几何体的长度。
- **ST_LongestLine** - 1.5.0 支持。返回两个几何体之间的最长线 (longest)。
- **ST_MakeEnvelope** - 1.5 支持。返回指定 SRID 和 SRS 的包络面。
- **ST_MaxDistance** - 1.5.0 支持。返回两个几何体之间的最大距离。
- **ST_ShortestLine** - 1.5.0 支持。返回两个几何体之间的最短线。
- **~=** - 1.5.0 支持。A 与 B 是否相等返回 TRUE 或 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 支持。Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- **ST_Collect** - 1.4.0 支持。返回 ST_MakeLine 返回的 GeometryCollection 或 Multi* geometry 从一组几何体。
- **ST_ContainsProperly** - 1.4.0 支持。Tests if every point of B lies in the interior of A
- **ST_GeoHash** - 1.4.0 支持。返回 GeoHash 值。
- **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 支持。Returns the portions of a geometry that lie in an elevation (Z) range.
- **ST_MakeLine** - 1.4.0 支持。返回 ST_MakeLine 返回的 GeometryCollection 或 Multi* geometry 从一组几何体。
- **ST_MinimumBoundingCircle** - 1.4.0 支持。Returns the smallest circle polygon that contains a geometry.
- **ST_Union** - Availability: 1.4.0 - ST_Union was enhanced. ST_Union(geometry array) 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 新增函数。将 GML 2 或 GML 3 格式转换为文本。
 - **ST_AsGeoJSON** - 1.3.4 新增函数。返回一个几何体或特征体的 GeoJSON 格式。
 - **ST_CurveToLine** - Availability: 1.3.0 将包含曲线的几何体转换为线性几何体。
 - **ST_LineToCurve** - Availability: 1.3.0 将线性几何体转换为曲线几何体。
 - **ST_SimplifyPreserveTopology** - 1.3.3 新增函数。返回一个简化的且有效的几何体表示，使用 Douglas-Peucker 算法。
-

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 3.6.0alpha1

2025/05/17

This version requires PostgreSQL 12-18beta1, GEOS 3.8 or higher, and Proj 6.1+. To take advantage of all features, GEOS 3.12+ is needed. To take advantage of all SFCGAL features, SFCGAL 2.1+ is needed.

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)

A.1.2 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)
