

PostGIS 3.7.0alpha1 ☒☒☒☒☒☒

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	安装与配置	6
2.2.1	安装	7
2.2.2	配置	7
2.2.3	环境变量	8
2.2.4	配置文件	10
2.2.5	PostGIS Extensions 简介	10
2.2.6	安装	12
2.2.7	配置	15
2.3	安装与配置	15
3	PostGIS Administration	16
3.1	Performance Tuning	16
3.1.1	Startup	16
3.1.2	Runtime	17
3.2	Configuring raster support	17
3.3	安装与配置	18
3.3.1	Spatially enable database using EXTENSION	18
3.3.2	Spatially enable database without using EXTENSION (discouraged)	18
3.4	Upgrading spatial databases	19
3.4.1	Soft upgrade	19
3.4.1.1	Soft upgrade using extensions	19
3.4.1.2	Soft upgrade without extensions	20
3.4.1.3	Hard upgrade	21

4 Data Management	23
4.1 GIS (GIS) 数据类型	23
4.1.1 OGC Geometry	23
4.1.1.1 Point	24
4.1.1.2 LineString	24
4.1.1.3 LinearRing	24
4.1.1.4 Polygon	24
4.1.1.5 MultiPoint	25
4.1.1.6 MultiLineString	25
4.1.1.7 MultiPolygon	25
4.1.1.8 GeometryCollection	25
4.1.1.9 PolyhedralSurface	25
4.1.1.10 Triangle	25
4.1.1.11 TIN	26
4.1.2 SQL-MM Part 3	26
4.1.2.1 CircularString	26
4.1.2.2 CompoundCurve	26
4.1.2.3 CurvePolygon	26
4.1.2.4 MultiCurve	27
4.1.2.5 MultiSurface	27
4.1.3 OpenGIS WKB 与 WKT	27
4.2 Geometry Data Type	28
4.2.1 OpenGIS WKB 与 WKT	28
4.3 PostGIS 数据类型	30
4.3.1 数据类型	31
4.3.2 PostGIS 数据类型	32
4.3.3 数据类型	33
4.3.4 数据类型 FAQ	33
4.4 Geometry Validation	34
4.4.1 Simple Geometry	34
4.4.2 Valid Geometry	36
4.4.3 Managing Validity	38
4.5 SPATIAL_REF_SYS 数据类型	38
4.5.1 SPATIAL_REF_SYS Table	39
4.5.2 SPATIAL_REF_SYS 数据类型	40
4.6 数据类型	41
4.6.1 数据类型	41
4.6.2 The GEOMETRY_COLUMNS VIEW	42
4.6.3 geometry_columns 数据类型	42

4.7	GIS (GIS) 数据库	44
4.7.1	SQL 数据库数据库数据库	45
4.7.2	shp2pgsql: ESRI shapefile 数据库数据库	45
4.8	数据库数据库	47
4.8.1	SQL 数据库数据库数据库数据库	47
4.8.2	数据库数据库	48
4.9	数据库数据库	48
4.9.1	GiST 数据库	49
4.9.2	GiST 数据库	50
4.9.3	GiST 数据库	51
4.9.4	数据库数据库数据库	52
5	Spatial Queries	53
5.1	Determining Spatial Relationships	53
5.1.1	Dimensionally Extended 9-Intersection Model	53
5.1.2	Named Spatial Relationships	55
5.1.3	General Spatial Relationships	56
5.2	Using Spatial Indexes	58
5.3	Examples of Spatial SQL	59
6	数据库数据库	62
6.1	数据库数据库数据库数据库数据库数据库	62
6.1.1	数据库数据库	62
6.1.2	数据库数据库	62
6.2	数据库数据库数据库数据库数据库数据库	63
6.3	数据库数据库数据库	63
7	PostGIS Reference	64
7.1	PostgreSQL PostGIS Geometry/Geography/Box 数据库	64
7.1.1	box2d	64
7.1.2	box3d	65
7.1.3	geometry	65
7.1.4	geometry_dump	66
7.1.5	geography	66
7.2	数据库数据库	67
7.2.1	AddGeometryColumn	67
7.2.2	DropGeometryColumn	69
7.2.3	DropGeometryTable	70
7.2.4	Find_SRID	71
7.2.5	Populate_Geometry_Columns	71

7.2.6	UpdateGeometrySRID	73
7.3	ST_GeomFromText (constructor)	74
7.3.1	ST_Collect	74
7.3.2	ST_LineFromMultiPoint	75
7.3.3	ST_MakeEnvelope	76
7.3.4	ST_MakeLine	77
7.3.5	ST_MakePoint	78
7.3.6	ST_MakePointM	79
7.3.7	ST_MakePolygon	81
7.3.8	ST_Point	83
7.3.9	ST_PointZ	84
7.3.10	ST_PointM	84
7.3.11	ST_PointZM	85
7.3.12	ST_Polygon	85
7.3.13	ST_TileEnvelope	86
7.3.14	ST_HexagonGrid	87
7.3.15	ST_Hexagon	90
7.3.16	ST_SquareGrid	91
7.3.17	ST_Square	92
7.3.18	ST_Letters	93
7.3.19	ST_MakeNurbsCurve	94
7.4	ST_Geometry (accessor)	95
7.4.1	GeometryType	95
7.4.2	ST_Boundary	96
7.4.3	ST_BoundingDiagonal	99
7.4.4	ST_CoordDim	99
7.4.5	ST_Dimension	100
7.4.6	ST_Dump	101
7.4.7	ST_DumpPoints	103
7.4.8	ST_DumpSegments	107
7.4.9	ST_DumpRings	109
7.4.10	ST_EndPoint	110
7.4.11	ST_Envelope	111
7.4.12	ST_ExteriorRing	113
7.4.13	ST_GeometryN	114
7.4.14	ST_GeometryType	116
7.4.15	ST_HasArc	117
7.4.16	ST_InteriorRingN	118
7.4.17	ST_NumCurves	119

7.4.18ST_CurveN	119
7.4.19ST_IsClosed	120
7.4.20ST_IsCollection	122
7.4.21ST_IsEmpty	123
7.4.22ST_IsPolygonCCW	124
7.4.23ST_IsPolygonCW	124
7.4.24ST_IsRing	125
7.4.25ST_IsSimple	126
7.4.26ST_M	127
7.4.27ST_MemSize	128
7.4.28ST_NDims	129
7.4.29ST_NPoints	129
7.4.30ST_NRings	130
7.4.31ST_NumGeometries	131
7.4.32ST_NumInteriorRings	132
7.4.33ST_NumInteriorRing	132
7.4.34ST_NumPatches	133
7.4.35ST_NumPoints	133
7.4.36ST_PatchN	134
7.4.37ST_PointN	135
7.4.38ST_Points	136
7.4.39ST_StartPoint	137
7.4.40ST_StartM	138
7.4.41ST_EndM	139
7.4.42ST_SetStartM	140
7.4.43ST_SetEndM	141
7.4.44ST_Summary	142
7.4.45ST_X	143
7.4.46ST_Y	144
7.4.47ST_Z	145
7.4.48ST_ZMFlag	146
7.4.49ST_HasZ	146
7.4.50ST_HasM	147
7.4.51ST_ControlPoints	148
7.4.52ST_Degree	149
7.4.53ST_Weights	149
7.4.54ST_Knots	150
7.4.55ST_NumControlPoints	151
7.4.56ST_NurbsCurveIsRational	151

7.4.57	ST_IsNurbsCurve	152
7.4.58	ST_Evaluate	153
7.4.59	ST_NurbsToLineString	154
7.5	ST_ (editor)	154
7.5.1	ST_AddPoint	154
7.5.2	ST_CollectionExtract	155
7.5.3	ST_CollectionHomogenize	156
7.5.4	ST_CurveToLine	158
7.5.5	ST_Scroll	160
7.5.6	ST_FlipCoordinates	161
7.5.7	ST_Force2D	162
7.5.8	ST_Force3D	163
7.5.9	ST_Force3DZ	163
7.5.10	ST_Force3DM	164
7.5.11	ST_Force4D	165
7.5.12	ST_ForceCollection	166
7.5.13	ST_ForceCurve	167
7.5.14	ST_ForcePolygonCCW	167
7.5.15	ST_ForcePolygonCW	168
7.5.16	ST_ForceSFS	168
7.5.17	ST_ForceRHR	169
7.5.18	ST_LineExtend	170
7.5.19	ST_LineToCurve	170
7.5.20	ST_Multi	171
7.5.21	ST_Normalize	172
7.5.22	ST_Project	173
7.5.23	ST_QuantizeCoordinates	174
7.5.24	ST_RemovePoint	176
7.5.25	ST_RemoveRepeatedPoints	176
7.5.26	ST_RemoveIrrelevantPointsForView	177
7.5.27	ST_RemoveSmallParts	180
7.5.28	ST_Reverse	181
7.5.29	ST_Segmentize	182
7.5.30	ST_SetPoint	183
7.5.31	ST_ShiftLongitude	184
7.5.32	ST_WrapX	185
7.5.33	ST_SnapToGrid	186
7.5.34	ST_Snap	188
7.5.35	ST_SwapOrdinates	191

7.6	Geometry Validation	192
7.6.1	ST_IsValid	192
7.6.2	ST_IsValidDetail	193
7.6.3	ST_IsValidReason	195
7.6.4	ST_MakeValid	196
7.7	Spatial Reference System Functions	201
7.7.1	ST_InverseTransformPipeline	201
7.7.2	ST_SetSRID	202
7.7.3	ST_SRID	203
7.7.4	ST_Transform	204
7.7.5	ST_TransformPipeline	206
7.7.6	postgis_srs_codes	208
7.7.7	postgis_srs	209
7.7.8	postgis_srs_all	209
7.7.9	postgis_srs_search	210
7.8	Geometry Input	211
7.8.1	Well-Known Text (WKT)	211
7.8.1.1	ST_BdPolyFromText	211
7.8.1.2	ST_BdMPolyFromText	212
7.8.1.3	ST_GeogFromText	213
7.8.1.4	ST_GeographyFromText	213
7.8.1.5	ST_GeomCollFromText	213
7.8.1.6	ST_GeomFromEWKT	214
7.8.1.7	ST_GeomFromMARC21	216
7.8.1.8	ST_GeometryFromText	218
7.8.1.9	ST_GeomFromText	218
7.8.1.10	ST_LineFromText	220
7.8.1.11	ST_MLineFromText	221
7.8.1.12	ST_MPointFromText	222
7.8.1.13	ST_MPolyFromText	222
7.8.1.14	ST_PointFromText	223
7.8.1.15	ST_PolygonFromText	224
7.8.1.16	ST_WKTToSQL	225
7.8.2	Well-Known Binary (WKB)	225
7.8.2.1	ST_GeogFromWKB	225
7.8.2.2	ST_GeomFromEWKB	226
7.8.2.3	ST_GeomFromWKB	227
7.8.2.4	ST_LineFromWKB	228
7.8.2.5	ST_LinestringFromWKB	229

7.8.2.6	ST_PointFromWKB	230
7.8.2.7	ST_WKBToSQL	231
7.8.3	Other Formats	231
7.8.3.1	ST_Box2dFromGeoHash	231
7.8.3.2	ST_GeomFromGeoHash	232
7.8.3.3	ST_GeomFromGML	233
7.8.3.4	ST_GeomFromGeoJSON	236
7.8.3.5	ST_GeomFromKML	237
7.8.3.6	ST_GeomFromTWKB	237
7.8.3.7	ST_GMLToSQL	238
7.8.3.8	ST_LineFromEncodedPolyline	238
7.8.3.9	ST_PointFromGeoHash	239
7.8.3.10	ST_FromFlatGeobufToTable	240
7.8.3.11	ST_FromFlatGeobuf	240
7.9	Geometry Output	241
7.9.1	Well-Known Text (WKT)	241
7.9.1.1	ST_AsEWKT	241
7.9.1.2	ST_AsText	242
7.9.2	Well-Known Binary (WKB)	243
7.9.2.1	ST_AsBinary	243
7.9.2.2	ST_AsEWKB	245
7.9.2.3	ST_AsHEXEWKB	246
7.9.3	Other Formats	247
7.9.3.1	ST_AsEncodedPolyline	247
7.9.3.2	ST_AsFlatGeobuf	248
7.9.3.3	ST_AsGeobuf	248
7.9.3.4	ST_AsGeoJSON	249
7.9.3.5	ST_AsGML	251
7.9.3.6	ST_AsKML	255
7.9.3.7	ST_AsLatLonText	256
7.9.3.8	ST_AsMARC21	257
7.9.3.9	ST_AsMVTGeom	260
7.9.3.10	ST_AsMVT	261
7.9.3.11	ST_AsSVG	263
7.9.3.12	ST_AsTWKB	264
7.9.3.13	ST_AsX3D	265
7.9.3.14	ST_GeoHash	268
7.10	几何运算符 (operator)	269
7.10.1	Bounding Box Operators	269

7.10.1.1	269
7.10.1.2(geometry,box2df)	270
7.10.1.3(box2df,geometry)	271
7.10.1.4(box2df,box2df)	272
7.10.1.5	273
7.10.1.6(geometry,gidx)	274
7.10.1.7(gidx,geometry)	275
7.10.1.8(gidx,gidx)	276
7.10.1.9<	276
7.10.1.10<	277
7.10.1.11>	278
7.10.1.12<	279
7.10.1.13<	280
7.10.1.14	280
7.10.1.15>	282
7.10.1.16@	283
7.10.1.17@ (geometry,box2df)	283
7.10.1.18@ (box2df,geometry)	284
7.10.1.19@ (box2df,box2df)	285
7.10.1.20>	286
7.10.1.21>	286
7.10.1.22	287
7.10.1.23(geometry,box2df)	288
7.10.1.24(box2df,geometry)	289
7.10.1.25(box2df,box2df)	289
7.10.1.26=	290
7.10.2 (operator)	291
7.10.2.1<->	291
7.10.2.2 =	293
7.10.2.3<#>	294
7.10.2.4<->>	295
7.11 Spatial Relationships	296
7.11.1 Topological Relationships	296
7.11.1.1 ST_3DIntersects	296
7.11.1.2 ST_Contains	297
7.11.1.3 ST_ContainsProperly	301
7.11.1.4 ST_CoveredBy	302
7.11.1.5 ST_Covers	303
7.11.1.6 ST_Crosses	305

7.11.1.7ST_Disjoint	307
7.11.1.8ST_Equals	308
7.11.1.9ST_Intersects	309
7.11.1.10ST_LineCrossingDirection	311
7.11.1.11ST_OrderingEquals	314
7.11.1.12ST_Overlaps	315
7.11.1.13ST_Relate	318
7.11.1.14ST_RelateMatch	320
7.11.1.15ST_Touches	321
7.11.1.16ST_Within	323
7.11.2Distance Relationships	325
7.11.2.1ST_3DDWithin	325
7.11.2.2ST_3DDFullyWithin	326
7.11.2.3ST_DFullyWithin	327
7.11.2.4ST_DWithin	328
7.11.2.5ST_PointInsideCircle	329
7.12Measurement Functions	330
7.12.1ST_Area	330
7.12.2ST_Azimuth	332
7.12.3ST_Angle	333
7.12.4ST_ClosestPoint	334
7.12.5ST_3DClosestPoint	336
7.12.6ST_Distance	337
7.12.7ST_3DDistance	339
7.12.8ST_DistanceSphere	340
7.12.9ST_DistanceSpheroid	341
7.12.10ST_FrechetDistance	342
7.12.11ST_HausdorffDistance	343
7.12.12ST_Length	345
7.12.13ST_Length2D	346
7.12.14ST_3DLength	346
7.12.15ST_LengthSpheroid	347
7.12.16ST_LongestLine	348
7.12.17ST_3DLongestLine	351
7.12.18ST_MaxDistance	352
7.12.19ST_3DMaxDistance	353
7.12.20ST_MinimumClearance	354
7.12.21ST_MinimumClearanceLine	354
7.12.22ST_Perimeter	355

7.12.2	ST_Perimeter2D	357
7.12.2	ST_3DPerimeter	357
7.12.2	ST_ShortestLine	358
7.12.2	ST_3DShortestLine	360
7.13	Overlay Functions	361
7.13.1	ST_ClipByBox2D	361
7.13.2	ST_Difference	362
7.13.3	ST_Intersection	363
7.13.4	ST_MemUnion	366
7.13.5	ST_Node	366
7.13.6	ST_Split	367
7.13.7	ST_Subdivide	369
7.13.8	ST_SymDifference	373
7.13.9	ST_UnaryUnion	374
7.13.10	ST_Union	375
7.14	Geometry Functions	378
7.14.1	ST_Buffer	378
7.14.2	ST_BuildArea	382
7.14.3	ST_Centroid	384
7.14.4	ST_ChaikinSmoothing	386
7.14.5	ST_CatmullRomSmoothing	387
7.14.6	ST_ConcaveHull	390
7.14.7	ST_ConvexHull	393
7.14.8	ST_DelaunayTriangles	395
7.14.9	ST_FilterByM	400
7.14.10	ST_GeneratePoints	401
7.14.11	ST_GeometricMedian	402
7.14.12	ST_LineMerge	403
7.14.13	ST_MaximumInscribedCircle	405
7.14.14	ST_LargestEmptyCircle	407
7.14.15	ST_MinimumBoundingCircle	409
7.14.16	ST_MinimumBoundingRadius	411
7.14.17	ST_OrientedEnvelope	411
7.14.18	ST_OffsetCurve	412
7.14.19	ST_PointOnSurface	416
7.14.20	ST_Polygonize	418
7.14.21	ST_ReducePrecision	420
7.14.22	ST_SharedPaths	421
7.14.23	ST_Simplify	423

7.14.25	ST_SimplifyPreserveTopology	425
7.14.26	ST_SimplifyPolygonHull	427
7.14.27	ST_SimplifyVW	430
7.14.28	ST_SetEffectiveArea	432
7.14.29	ST_TriangulatePolygon	433
7.14.30	ST_VoronoiLines	435
7.14.31	ST_VoronoiPolygons	436
7.15	Coverages	438
7.15.1	ST_CoverageInvalidEdges	438
7.15.2	ST_CoverageClean	440
7.15.3	ST_CoverageSimplify	442
7.15.4	ST_CoverageEdges	444
7.15.5	ST_CoverageUnion	445
7.16	Affine Transformations	446
7.16.1	ST_Affine	446
7.16.2	ST_Rotate	448
7.16.3	ST_RotateX	449
7.16.4	ST_RotateY	450
7.16.5	ST_RotateZ	451
7.16.6	ST_Scale	452
7.16.7	ST_Translate	453
7.16.8	ST_TransScale	454
7.17	Clustering Functions	455
7.17.1	ST_ClusterDBSCAN	455
7.17.2	ST_ClusterIntersecting	458
7.17.3	ST_ClusterIntersectingWin	458
7.17.4	ST_ClusterRelateWin	459
7.17.5	ST_MinimumSpanningTree	460
7.17.6	ST_ClusterKMeans	461
7.17.7	ST_ClusterWithin	463
7.17.8	ST_ClusterWithinWin	464
7.18	Bounding Box Functions	465
7.18.1	Box2D	465
7.18.2	Box3D	466
7.18.3	ST_EstimatedExtent	467
7.18.4	ST_Expand	468
7.18.5	ST_Extent	469
7.18.6	ST_3DExtent	470
7.18.7	ST_MakeBox2D	471

7.18.8ST_3DMakeBox	472
7.18.9ST_XMax	472
7.18.10T_XMin	473
7.18.11ST_YMax	474
7.18.12ST_YMin	475
7.18.13ST_ZMax	476
7.18.14ST_ZMin	477
7.18.15ST_XSize	478
7.18.16T_YSize	479
7.18.17ST_ZSize	480
7.18.18T_MMin	481
7.18.19ST_MMax	482
7.18.20T_MSize	482
7.19 几何函数 (Linear Referencing)	483
7.19.1ST_LineInterpolatePoint	483
7.19.2ST_3DLineInterpolatePoint	484
7.19.3ST_LineInterpolatePoints	485
7.19.4ST_LineLocatePoint	486
7.19.5ST_LineSubstring	487
7.19.6ST_LocateAlong	490
7.19.7ST_LocateBetween	491
7.19.8ST_LocateBetweenElevations	492
7.19.9ST_InterpolatePoint	493
7.19.10T_3DInterpolatePoint	494
7.19.11ST_AddMeasure	495
7.20Trajectory Functions	496
7.20.1ST_IsValidTrajectory	496
7.20.2ST_ClosestPointOfApproach	496
7.20.3ST_DistanceCPA	497
7.20.4ST_CPAWithin	498
7.21Version Functions	499
7.21.1PostGIS_Extensions_Upgrade	499
7.21.2PostGIS_Full_Version	500
7.21.3PostGIS_GEOS_Version	500
7.21.4PostGIS_GEOS_Compiled_Version	501
7.21.5PostGIS_Liblwgeom_Version	501
7.21.6PostGIS_LibXML_Version	502
7.21.7PostGIS_LibJSON_Version	502
7.21.8PostGIS_Lib_Build_Date	503

7.21.9	PostGIS_Lib_Version	503
7.21.10	PostGIS_PROJ_Version	504
7.21.11	PostGIS_PROJ_Compiled_Version	504
7.21.12	PostGIS_Wagyu_Version	505
7.21.13	PostGIS_Scripts_Build_Date	505
7.21.14	PostGIS_Scripts_Installed	506
7.21.15	PostGIS_Scripts_Released	507
7.21.16	PostGIS_Version	507
7.22	PostGIS GUC(Grand Unified Custom Variable)	508
7.22.1	postgis.gdal_datapath	508
7.22.2	postgis.gdal_enabled_drivers	509
7.22.3	postgis.enable_outdb_rasters	510
7.22.4	postgis.gdal_vsi_options	511
7.22.5	postgis.gdal_cpl_debug	512
7.23	Troubleshooting Functions	512
7.23.1	PostGIS_AddBBox	512
7.23.2	PostGIS_DropBBox	513
7.23.3	PostGIS_HasBBox	513

8 SFCGAL Functions Reference 515

8.1	SFCGAL Management Functions	515
8.1.1	postgis_sfcgal_version	515
8.1.2	postgis_sfcgal_full_version	515
8.2	SFCGAL Accessors and Setters	516
8.2.1	CG_ForceLHR	516
8.2.2	CG_IsPlanar	516
8.2.3	CG_IsSolid	517
8.2.4	CG_MakeSolid	517
8.2.5	CG_Orientation	518
8.2.6	CG_Area	518
8.2.7	CG_3DArea	519
8.2.8	CG_Volume	519
8.2.9	ST_ForceLHR	520
8.2.10	ST_IsPlanar	521
8.2.11	ST_IsSolid	521
8.2.12	ST_MakeSolid	522
8.2.13	ST_Orientation	522
8.2.14	ST_3DArea	523
8.2.15	ST_Volume	524

8.3	SFCGAL Processing and Relationship Functions	525
8.3.1	CG_Intersection	525
8.3.2	CG_Intersects	526
8.3.3	CG_3DIntersects	526
8.3.4	CG_Difference	527
8.3.5	ST_3DDifference	528
8.3.6	CG_3DDifference	529
8.3.7	CG_Distance	530
8.3.8	CG_3DDistance	530
8.3.9	ST_3DConvexHull	531
8.3.10	CG_3DConvexHull	532
8.3.11	ST_3DIntersection	533
8.3.12	CG_3DIntersection	534
8.3.13	CG_Union	536
8.3.14	ST_3DUnion	537
8.3.15	CG_3DUnion	537
8.3.16	ST_AlphaShape	538
8.3.17	CG_AlphaShape	539
8.3.18	CG_ApproxConvexPartition	542
8.3.19	ST_ApproximateMedialAxis	544
8.3.20	CG_ApproximateMedialAxis	544
8.3.21	CG_GenerateFlatRoof	546
8.3.22	CG_GenerateHippedRoof	546
8.3.23	CG_GenerateGableRoof	547
8.3.24	CG_GenerateSkillionRoof	548
8.3.25	CG_GenerateRoof	549
8.3.26	CG_PolygonRepair	550
8.3.27	ST_ConstrainedDelaunayTriangles	551
8.3.28	CG_ConstrainedDelaunayTriangles	551
8.3.29	ST_Extrude	552
8.3.30	CG_Extrude	553
8.3.31	CG_ExtrudeStraightSkeleton	554
8.3.32	CG_GreeneApproxConvexPartition	555
8.3.33	ST_MinkowskiSum	557
8.3.34	CG_MinkowskiSum	557
8.3.35	ST_OptimalAlphaShape	559
8.3.36	CG_OptimalAlphaShape	560
8.3.37	CG_OptimalConvexPartition	562
8.3.38	CG_StraightSkeleton	563

8.3.39	ST_StraightSkeleton	565
8.3.40	ST_Tessellate	566
8.3.41	CG_Tessellate	567
8.3.42	CG_Triangulate	569
8.3.43	CG_Visibility	570
8.3.44	CG_YMonotonePartition	571
8.3.45	CG_StraightSkeletonPartition	572
8.3.46	CG_3DBuffer	573
8.3.47	CG_Rotate	575
8.3.48	CG_2DRotate	576
8.3.49	CG_3DRotate	576
8.3.50	CG_RotateX	577
8.3.51	CG_RotateY	578
8.3.52	CG_RotateZ	578
8.3.53	CG_Scale	579
8.3.54	CG_3DScale	579
8.3.55	CG_3DScaleAroundCenter	580
8.3.56	CG_Translate	581
8.3.57	CG_3DTranslate	581
8.3.58	CG_Simplify	582
8.3.59	CG_3DAlphaWrapping	585
8.3.60	CG_NurbsCurveInterpolate	587
8.3.61	CG_NurbsCurveApproximate	588
8.3.62	CG_NurbsCurveDerivative	589
9	拓扑 (topology)	591
9.1	Topology Primitive Tables	591
9.2	拓扑操作	592
9.2.1	getfaceedges_returntype	592
9.2.2	TopoGeometry	592
9.2.3	validatetopology_returntype	593
9.3	拓扑对象	593
9.3.1	TopoElement	593
9.3.2	TopoElementArray	594
9.4	拓扑 TopoGeometry 操作	595
9.4.1	AddTopoGeometryColumn	595
9.4.2	RenameTopoGeometryColumn	596
9.4.3	DropTopology	596
9.4.4	RenameTopology	597

9.4.5	DropTopoGeometryColumn	597
9.4.6	FixCorruptTopoGeometryColumn	598
9.4.7	Populate_Topology_Layer	599
9.4.8	TopologySummary	600
9.4.9	ValidateTopology	600
9.4.10	ValidateTopologyRelation	603
9.4.11	ValidateTopologyPrecision	603
9.4.12	MakeTopologyPrecise	604
9.4.13	FindTopology	605
9.4.14	FindLayer	605
9.4.15	TotalTopologySize	606
9.4.16	UpgradeTopology	606
9.5	Topology Statistics Management	607
9.6	拓扑管理	607
9.6.1	CreateTopology	607
9.6.2	CopyTopology	608
9.6.3	ST_InitTopoGeo	609
9.6.4	ST_CreateTopoGeo	610
9.6.5	TopoGeo_AddPoint	611
9.6.6	TopoGeo_AddLineString	611
9.6.7	TopoGeo_AddPolygon	612
9.6.8	TopoGeo_LoadGeometry	612
9.7	拓扑操作	613
9.7.1	ST_AddIsoNode	613
9.7.2	ST_AddIsoEdge	614
9.7.3	ST_AddEdgeNewFaces	614
9.7.4	ST_AddEdgeModFace	615
9.7.5	ST_RemEdgeNewFace	615
9.7.6	ST_RemEdgeModFace	616
9.7.7	ST_ChangeEdgeGeom	617
9.7.8	ST_ModEdgeSplit	618
9.7.9	ST_ModEdgeHeal	618
9.7.10	ST_NewEdgeHeal	619
9.7.11	ST_MoveIsoNode	619
9.7.12	ST_NewEdgesSplit	620
9.7.13	ST_RemoveIsoNode	621
9.7.14	ST_RemoveIsoEdge	622
9.8	拓扑查询	622
9.8.1	GetEdgeByPoint	622

9.8.2	GetFaceByPoint	623
9.8.3	GetFaceContainingPoint	624
9.8.4	GetNodeByPoint	625
9.8.5	GetTopologyID	626
9.8.6	GetTopologySRID	626
9.8.7	GetTopologyName	627
9.8.8	ST_GetFaceEdges	627
9.8.9	ST_GetFaceGeometry	628
9.8.10	GetRingEdges	629
9.8.11	GetNodeEdges	629
9.8.12	FindVertexSegmentPairsBelowDistance	630
9.9	拓扑几何	630
9.9.1	Polygonize	630
9.9.2	AddNode	631
9.9.3	AddEdge	632
9.9.4	AddFace	633
9.9.5	ST_Simplify	635
9.9.6	RemoveUnusedPrimitives	635
9.10	TopoGeometry 对象	636
9.10.1	CreateTopoGeom	636
9.10.2	toTopoGeom	637
9.10.3	TopoElementArray_Agg	639
9.10.4	TopoElement	639
9.11	TopoGeometry 函数	640
9.11.1	clearTopoGeom	640
9.11.2	TopoGeom_addElement	641
9.11.3	TopoGeom_remElement	641
9.11.4	TopoGeom_addTopoGeom	642
9.11.5	toTopoGeom	642
9.12	TopoGeometry 属性	643
9.12.1	GetTopoGeomElementArray	643
9.12.2	GetTopoGeomElements	643
9.12.3	ST_SRID	644
9.13	TopoGeometry 转换	644
9.13.1	AsGML	644
9.13.2	AsTopoJSON	647
9.14	拓扑几何操作	648
9.14.1	Equals	648
9.14.2	Intersects	649
9.15	Importing and exporting Topologies	650
9.15.1	Using the Topology exporter	650
9.15.2	Using the Topology importer	651

10	栅格数据类型, 函数	652
10.1	栅格数据类型	652
10.1.1	raster2pgsql 函数	652
10.1.1.1	Example Usage	652
10.1.1.2	raster2pgsql options	653
10.1.2	PostGIS 栅格数据类型	655
10.1.3	Using "out db" cloud rasters	655
10.2	栅格函数	656
10.2.1	栅格函数	656
10.2.2	栅格函数	657
10.3	PostGIS 栅格数据类型与各种语言接口	658
10.3.1	栅格数据类型 ST_AsPNG 函数 PHP 接口	659
10.3.2	栅格数据类型 ST_AsPNG 函数 ASP.NET C# 接口	659
10.3.3	栅格数据类型 ST_AsPNG 函数 Java 接口	661
10.3.4	PLPython 函数 SQL 接口	662
10.3.5	PSQL 函数	663
11	栅格函数	664
11.1	栅格函数	665
11.1.1	geomval	665
11.1.2	addbandarg	665
11.1.3	rastbandarg	665
11.1.4	raster	666
11.1.5	reclassarg	666
11.1.6	summarystats	667
11.1.7	unionarg	668
11.2	栅格函数	668
11.2.1	AddRasterConstraints	668
11.2.2	DropRasterConstraints	670
11.2.3	AddOverviewConstraints	671
11.2.4	DropOverviewConstraints	672
11.2.5	PostGIS_GDAL_Version	672
11.2.6	PostGIS_Raster_Lib_Build_Date	673
11.2.7	PostGIS_Raster_Lib_Version	673
11.2.8	ST_GDALDrivers	674
11.2.9	UpdateRasterSRID	679
11.2.10	ST_CreateOverview	679
11.3	栅格函数 (constructor)	680
11.3.1	ST_AddBand	680

11.3.2	ST_AsRaster	683
11.3.3	ST_AsRasterAgg	685
11.3.4	ST_Band	686
11.3.5	ST_MakeEmptyCoverage	688
11.3.6	ST_MakeEmptyRaster	689
11.3.7	ST_Tile	690
11.3.8	ST_Retile	692
11.3.9	ST_FromGDALRaster	693
11.4	ST (accessor)	694
11.4.1	ST_GeoReference	694
11.4.2	ST_Height	695
11.4.3	ST_IsEmpty	695
11.4.4	ST_MemSize	696
11.4.5	ST_MetaData	697
11.4.6	ST_NumBands	697
11.4.7	ST_PixelHeight	698
11.4.8	ST_PixelWidth	699
11.4.9	ST_ScaleX	700
11.4.10	ST_ScaleY	701
11.4.11	ST_RasterToWorldCoord	701
11.4.12	ST_RasterToWorldCoordX	702
11.4.13	ST_RasterToWorldCoordY	703
11.4.14	ST_Rotation	704
11.4.15	ST_SkewX	705
11.4.16	ST_SkewY	706
11.4.17	ST_SRID	706
11.4.18	ST_Summary	707
11.4.19	ST_UpperLeftX	708
11.4.20	ST_UpperLeftY	708
11.4.21	ST_Width	709
11.4.22	ST_WorldToRasterCoord	709
11.4.23	ST_WorldToRasterCoordX	710
11.4.24	ST_WorldToRasterCoordY	711
11.5	ST_Band (accessor)	712
11.5.1	ST_BandMetaData	712
11.5.2	ST_BandNoDataValue	713
11.5.3	ST_BandIsNoData	714
11.5.4	ST_BandPath	715
11.5.5	ST_BandFileSize	716

11.5.6	ST_BandFileTimestamp	716
11.5.7	ST_BandPixelType	717
11.5.8	ST_MinPossibleValue	717
11.5.9	ST_HasNoBand	718
11.6	ST_PixelAsPolygon (setter)	719
11.6.1	ST_PixelAsPolygon	719
11.6.2	ST_PixelAsPolygons	719
11.6.3	ST_PixelAsPoint	720
11.6.4	ST_PixelAsPoints	721
11.6.5	ST_PixelAsCentroid	722
11.6.6	ST_PixelAsCentroids	723
11.6.7	ST_Value	724
11.6.8	ST_NearestValue	727
11.6.9	ST_SetZ	729
11.6.10	ST_SetM	730
11.6.11	ST_Neighborhood	731
11.6.12	ST_SetValue	733
11.6.13	ST_SetValues	734
11.6.14	ST_DumpValues	742
11.6.15	ST_PixelOfValue	743
11.7	ST_SetGeoReference	745
11.7.1	ST_SetGeoReference	745
11.7.2	ST_SetRotation	746
11.7.3	ST_SetScale	747
11.7.4	ST_SetSkew	748
11.7.5	ST_SetSRID	749
11.7.6	ST_SetUpperLeft	749
11.7.7	ST_Resample	750
11.7.8	ST_Rescale	751
11.7.9	ST_Reskew	753
11.7.10	ST_SnapToGrid	754
11.7.11	ST_Resize	755
11.7.12	ST_Transform	757
11.8	ST_SetBandNoDataValue	759
11.8.1	ST_SetBandNoDataValue	759
11.8.2	ST_SetBandIsNoData	760
11.8.3	ST_SetBandPath	762
11.8.4	ST_SetBandIndex	763
11.9	ST_SetBandNoDataValue	764

11.9.1	ST_Count	764
11.9.2	ST_CountAgg	765
11.9.3	ST_Histogram	766
11.9.4	ST_Quantile	768
11.9.5	ST_SummaryStats	770
11.9.6	ST_SummaryStatsAgg	772
11.9.7	ST_ValueCount	773
11.10	Raster Inputs	776
11.10.1	ST_RastFromWKB	776
11.10.2	ST_RastFromHexWKB	777
11.11	ST_AsBinary/ST_AsWKB	777
11.11.1	ST_AsHexWKB	778
11.11.2	ST_AsGDALRaster	779
11.11.3	ST_AsJPEG	780
11.11.4	ST_AsPNG	781
11.11.5	ST_AsTIFF	782
11.12	ST_Clip	783
11.12.1	ST_ColorMap	786
11.12.2	ST_Grayscale	789
11.12.3	ST_Intersection	791
11.12.4	ST_MapAlgebra (callback function version)	793
11.12.5	ST_MapAlgebra (expression version)	799
11.12.6	ST_MapAlgebraExpr	802
11.12.7	ST_MapAlgebraExpr	804
11.12.8	ST_MapAlgebraFct	809
11.12.9	ST_MapAlgebraFct	812
11.12.10	ST_MapAlgebraFctNgb	817
11.12.11	ST_Reclass	819
11.12.12	ST_ReclassExact	821
11.12.13	ST_Union	823
11.13	ST_Distinct4ma	824
11.13.1	ST_InvDistWeight4ma	825
11.13.2	ST_Max4ma	826
11.13.3	ST_Mean4ma	827
11.13.4	ST_Min4ma	829
11.13.5	ST_MinDist4ma	830

11.13.	\$T_Range4ma	831
11.13.	\$T_StdDev4ma	832
11.13.	\$T_Sum4ma	833
11.14.	函数参考	834
11.14.	\$T_Aspect	834
11.14.	\$T_HillShade	836
11.14.	\$T_Roughness	837
11.14.	\$T_Slope	838
11.14.	\$T_TPI	840
11.14.	\$T_TRI	841
11.14.	\$T_InterpolateRaster	842
11.14.	\$T_Contour	843
11.15.	函数参考	844
11.15.	\$Box3D	844
11.15.	\$T_ConvexHull	845
11.15.	\$T_DumpAsPolygons	846
11.15.	\$T_Envelope	847
11.15.	\$T_MinConvexHull	847
11.15.	\$T_Polygon	849
11.15.	\$T_IntersectionFractions	850
11.16.	函数参考	851
11.16.	\$&&	851
11.16.	\$<	852
11.16.	\$>	852
11.16.	\$#	853
11.16.	\$@	854
11.16.	\$=	854
11.16.	\$%	855
11.17.	函数参考	855
11.17.	\$T_Contains	855
11.17.	\$T_ContainsProperly	857
11.17.	\$T_Covers	857
11.17.	\$T_CoveredBy	858
11.17.	\$T_Disjoint	859
11.17.	\$T_Intersects	860
11.17.	\$T_Overlaps	861
11.17.	\$T_Touches	862
11.17.	\$T_SameAlignment	863
11.17.	\$T_NotSameAlignmentReason	864

11.17.\$T_Within	865
11.17.\$T_DWithin	866
11.17.\$T_DFullyWithin	867
11.18.Raster Tips	868
11.18.Out-DB Rasters	868
11.18.1.Directory containing many files	868
11.18.1.Maximum Number of Open Files	868
11.18.1.2.Maximum number of open files for the entire system	869
11.18.1.2.Maximum number of open files per process	869
12 PostGIS Extras	871
13 PostGIS Special Functions Index	872
13.1 PostGIS Aggregate Functions	872
13.2 PostGIS Window Functions	873
13.3 PostGIS SQL-MM Compliant Functions	874
13.4 PostGIS Geography Support Functions	878
13.5 PostGIS Raster Support Functions	879
13.6 PostGIS Geometry / Geography / Raster Dump Functions	885
13.7 PostGIS Box Functions	885
13.8 PostGIS Functions that support 3D	887
13.9 PostGIS Curved Geometry Support Functions	893
13.10 PostGIS Polyhedral Surface Support Functions	896
13.1 PostGIS Function Support Matrix	900
13.1 New, Enhanced or changed PostGIS Functions	920
13.12.PostGIS Functions new or enhanced in 3.7	920
13.12.PostGIS Functions new or enhanced in 3.6	922
13.12.PostGIS Functions new or enhanced in 3.5	923
13.12.PostGIS Functions new or enhanced in 3.4	925
13.12.PostGIS Functions new or enhanced in 3.3	926
13.12.PostGIS Functions new or enhanced in 3.2	927
13.12.PostGIS Functions new or enhanced in 3.1	929
13.12.PostGIS Functions new or enhanced in 3.0	930
13.12.PostGIS Functions new or enhanced in 2.5	932
13.12.PostGIS Functions new or enhanced in 2.4	933
13.12.PostGIS Functions new or enhanced in 2.3	934
13.12.PostGIS Functions new or enhanced in 2.2	936
13.12.PostGIS Functions new or enhanced in 2.1	939
13.12.PostGIS Functions new or enhanced in 2.0	945
13.12.PostGIS Functions new or enhanced in 1.5	956
13.12.PostGIS Functions new or enhanced in 1.4	957
13.12.PostGIS Functions new or enhanced in 1.3	958

14 Reporting Problems	959
14.1 Reporting Software Bugs	959
14.2 Reporting Documentation Issues	959
A Appendix	961
A.1 PostGIS 3.7.0alpha1	961
A.1.1 Breaking Changes	961
A.1.2 New Features	961
A.1.3 Enhancements	962
A.1.4 Bug Fixes	963

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 3.7.0alpha1 发布。



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。
<https://postgis.net>

Chapter 1

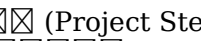
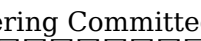
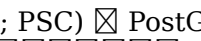


PostGIS is a spatial extension for the PostgreSQL relational database that was created by Refractions Research Inc, as a spatial database technology research project. Refractions is a GIS and database consulting company in Victoria, British Columbia, Canada, specializing in data integration and custom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

Laurențiu Nicola Bug fixes, maintenance, stability, and performance

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 的 CMake 支持, 支持, 支持 API 支持

Kevin Neufeld PSC 成员。支持, 支持, PostGIS 支持, PostGIS 支持

Dave Blasby PostGIS 支持。支持, 支持

Jeff Lounsbury shapefile 支持/支持。支持 PostGIS 支持

Mark Leslie 支持。支持, shapefile GUI 支持

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

David Zwarg 支持 (支持) 支持

1.4

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

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

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

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

1. **Geographic Information Systems (GIS):** GIS is a system that captures, stores, analyzes, and displays spatial data. It is used in various fields such as urban planning, environmental management, and transportation. GIS software like ArcGIS and QGIS are commonly used.

PostGIS has experimented with community fundraising for features in the past. We used [PledgeBank](#) and we got two successful campaigns out of it.

postgistopology - 10 2.0.0 toTopGeometry 250.

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

☐☐☐☐☐☐☐☐ The GEOS geometry operations library

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

The PROJ cartographic projection library

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

Chapter 2

PostGIS

PostGIS is a spatial database engine for PostgreSQL.

2.1

PostGIS is a spatial database engine for PostgreSQL.

```
tar -xvzf postgis-3.7.0alpha1.tar.gz
cd postgis-3.7.0alpha1
./configure
make
make install
```

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

2.2

Note

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

Pre-Built Packages for various OS are listed in [PostGIS Pre-built Packages](#) [Stackbuilder](#) [PostGIS Windows download site](#) [very bleeding-edge windows experimental builds](#) [PostGIS](#).

1~2 [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 3.7.0alpha1 *requires* full PostgreSQL server headers access in order to compile. It can be built against PostgreSQL versions 14 - 18. Earlier versions of PostgreSQL are *not* supported.

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

Note

GEOS 依赖于 PostgreSQL 依赖于 C++ 编译选项。编译选项如下。



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

编译选项如下 C++ 编译选项。编译选项如下 (编译选项如下) 编译选项 PostgreSQL 编译选项。

PostGIS 编译选项如下。编译选项如下。

2.2.1 安装

PostGIS 编译选项如下 <https://download.osgeo.org/postgis/source/postgis-3.7.0alpha1.tar.gz> 编译选项如下。

```
wget https://download.osgeo.org/postgis/source/postgis-3.7.0alpha1.tar.gz
tar -xvzf postgis-3.7.0alpha1.tar.gz
cd postgis-3.7.0alpha1
```

编译选项如下 postgis-3.7.0alpha1 (编译选项如下) 编译选项如下。

编译选项如下, [svn](http://svn.osgeo.org/postgis/trunk/) 编译选项 <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
```

编译选项如下 postgis-3.7.0alpha1 编译选项如下。

```
./configure
```

2.2.2 编译选项

PostGIS 编译选项如下。

编译选项

- PostgreSQL 14 - 18. A complete installation of PostgreSQL (including server headers) is required. PostgreSQL is available from <https://www.postgresql.org> 18 .
For a full PostgreSQL / PostGIS support matrix and PostGIS/GEOS support matrix refer to <https://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS>
- GNU C 编译器 (gcc). PostGIS 编译选项如下 ANSI C 编译选项如下 gcc 编译选项如下。
- GNU Make(gmake 或 make). 编译选项如下 GNU make 或 make 编译选项如下。make -v 编译选项如下。编译选项 make 或 PostGIS Makefile 编译选项如下。
- Proj reprojection library. Proj 6.1 or above is required. The Proj library is used to provide coordinate reprojection support within PostGIS. Proj is available for download from <https://proj.org/> .
- GEOS geometry library, version 3.8.0 or greater, but GEOS 3.14+ is required to take full advantage of all the new functions and features. GEOS is available for download from <https://libgeos.org> .

- LibXML2, version 2.5.x or higher. LibXML2 is currently used in some imports functions (ST_GeomFromGML and ST_GeomFromKML). LibXML2 is available for download from <https://gitlab.gnome.org/GNOME/libxml2/-/releases>.
- JSON-C 0.9 或更高版本。JSON-C 用于 ST_GeomFromGeoJson 和 GeoJSON 的导入。JSON-C 可在 <https://github.com/json-c/json-c/releases/> 下载。
- The GNU Multiple Precision Arithmetic Library (libgmp). Used to control precision of math operations in PostGIS Topology. GMP is available for download from <https://gmplib.org/>.
- GDAL, version 3+ is preferred. This is required for raster support. <https://gdal.org/download.html>.
- PostgreSQL 12.0 或更高版本。PostgreSQL 12.0 或更高版本。可在 <http://trac.osgeo.org/postgis/ticket/635> 查看。

安装依赖

- 第 2.1 节 安装依赖。安装依赖。
- shapefile 到 shp2pgsql-gui 需要 GTK (GTK+2.0, 2.8+ 或更高)。 <http://www.gtk.org/>.
- SFCGAL, 1.4.1 或更高版本是必需的, 2.3+ 是需要的, 以便使用所有功能。SFCGAL 可以用于提供额外的 2D 和 3D 高级分析函数到 PostGIS 第 8 章。 <https://sfcgal.gitlab.io/sfcgal/>.
- 要启用 ST_AsMVT protobuf-c 库 1.1.0 或更高版本 (用于使用) 和 protoc-c 编译器 (用于构建) 是必需的。Also, pkg-config 是必需的, 以验证 protobuf-c 的最小版本。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). 可在 <http://cunit.sourceforge.net/> 下载。
- DocBook(xsltproc) 可在 <http://www.docbook.org/> 下载。
- DBLatex(dblatex) 可在 <http://dblatex.sourceforge.net/> 下载。
- GraphicsMagick (gm convert) is required to generate the images used in the documentation. If GraphicsMagick is unavailable, ImageMagick (convert or magick convert) provides the same command-line syntax. GraphicsMagick is available from <http://www.graphicsmagick.org/> and ImageMagick from <https://imagemagick.org/>.

2.2.3 安装

安装 PostGIS 3.7.0alpha1 需要安装 Makefile 和 Makefile。

./configure

安装 PostGIS 3.7.0alpha1, 需要安装 PostGIS 3.7.0alpha1 的 Makefile 和 Makefile。./configure 需要安装 PostGIS 3.7.0alpha1 的 Makefile 和 Makefile。

安装 PostGIS 3.7.0alpha1 需要安装 Makefile 和 Makefile。安装 --help 或 --help=short 安装。

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



XXXXXXXXXXXXXXXXXXXX. XXXX PostgreSQL XXXXXXXXXXXXXXXXXXXXXXX.
XXXXXXXXXXXX <http://trac.osgeo.org/postgis/ticket/635> XXXXXXXX.

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

Note

PostGIS 的 SVN 仓库地址, 请前往该地址。

**./autogen.sh**

运行 **configure** 脚本, 该脚本会生成 PostGIS 的 Makefile 文件。

运行 `tar -xzf PostGIS-3.7.0alpha1.tar.gz` 解压 PostGIS 的 tar 包, 然后运行 **configure** 脚本, 最后运行 **./autogen.sh** 脚本。

2.2.4 编译

运行 Makefile 文件中的 `PostGIS` 目标。

make

运行 `make` 命令, 会生成 `PostGIS was built successfully. Ready to install.` 消息。

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 comment files 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

The **make cheatsheets** target generates html cheat sheets suitable for quick reference or for student handouts. This requires xsltproc to build and will generate 4 files in `doc` folder `topology_cheatsheet.html`, `raster_cheatsheet.html`, `postgis_cheatsheet.html`

html 和 pdf 格式的 PostGIS / PostgreSQL 学习指南。

make cheatsheets

2.2.5 PostGIS Extensions 扩展

The PostGIS extensions are built and installed automatically when PostgreSQL extension support is available.

运行 `make` 命令, 会生成 `function descriptions` 文件。docbook 格式的文件, 请前往该地址:

make comments

运行 `tar -xzf PostGIS-3.7.0alpha1.tar.gz` 解压 PostGIS 的 tar 包, 然后运行 `make comments` 命令。

运行 PostgreSQL 9.1 的 `extensions` 脚本, 该脚本会生成 `extensions` 文件。

```
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 extensions 安装 PostGIS 安装。

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

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

psql 安装。

```
SELECT name, default_version, installed_version
FROM pg_available_extensions
WHERE name LIKE 'postgis%'
ORDER BY name;
```

name	default_version	installed_version
postgis	3.7.0alpha1	3.7.0alpha1
postgis_raster	3.7.0alpha1	3.7.0alpha1
postgis_sfcgal	3.7.0alpha1	
postgis_topology	3.7.0alpha1	

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

extension 安装 pgAdmin extension 安装 sql 安装 postgis extension 安装:

```
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_raster;
CREATE EXTENSION postgis_sfcgal;
CREATE EXTENSION postgis_topology;
```

PSQL 安装, 安装。

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

List of installed extensions

```
-[ RECORD 1 ]-----
Name          | postgis
Version       | 3.7.0alpha1
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_topology
Version		3.7.0alpha1
Schema		topology
Description		PostGIS topology spatial types and functions



Warning
Extension tables `spatial_ref_sys`, `layer`, `topology` can not be explicitly backed up. They can only be backed up when the respective `postgis` or `postgis_topology` extension is backed up, which only happens when you back up the whole database. Only `sr_id` records not packaged with PostGIS are captured in backups, so don't change the entries we ship and expect the modifications to persist. Put in a ticket if you find an issue. The structures of extension tables are never backed up since they are created with `CREATE EXTENSION` and assumed to be the same for a given version of an extension. These behaviors are built into the current PostgreSQL extension model.

PostGIS 3.7.0alpha1, 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;
```

2.2.6

PostGIS, make check.

make check

PostgreSQL, make check.



Note
PostgreSQL, GEOS, Proj4, LD_LIBRARY_PATH.



Caution
make check PATH PGPORT. PostgreSQL, --with-pgconfig, PostgreSQL, PATH.

Note

Sandboxed build accounts that are not PostgreSQL superusers can delegate database ownership during the regression cycle by exporting `POSTGIS_REGRESS_DB_OWNER`. The harness will create the temporary regression database owned by the nominated role while continuing to connect using the less privileged account. Combine this with `POSTGIS_REGRESS_ROLE_EXT_CREATOR` when the extension creation role must differ from the database owner.

These variables allow automated environments to exercise the full upgrade and extension install paths without promoting the calling account to superuser, provided the target PostgreSQL instance permits extension installation by those delegate roles.

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
```

XXXXXXXXXXXXXXXXXXXX:

```
===== dropping database "contrib_regression" =====
DROP DATABASE
===== creating database "contrib_regression" =====
CREATE DATABASE
ALTER DATABASE
===== installing fuzzystmatch =====
CREATE EXTENSION
===== installing postgis =====
CREATE EXTENSION
===== running regression test queries =====
```

```
test test-normalize_address ... ok
```

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

2.2.7 安装

PostGIS 安装指南。

make install

`make --prefix` 安装 PostGIS 到指定目录。

- `loader` 安装到 `[prefix]/bin` 目录。
- `postgis.sql` SQL 脚本安装到 `[prefix]/share/contrib` 目录。
- PostGIS 库安装到 `[prefix]/lib` 目录。

安装 `postgis_comments.sql`, `raster_comments.sql` 和 `make comments` 脚本，`make comments` 脚本会安装 `sql` 脚本。

make comments-install



Note

xsitproc 安装 `postgis_comments.sql`, `raster_comments.sql`, `topology_comments.sql` 脚本。

2.3 验证安装

验证安装是否成功。

1. PostgreSQL 14 安装成功。安装 PostgreSQL 14 到指定目录。 (Linux) 安装 PostgreSQL 14 到指定目录，安装 PostgreSQL 14 到指定目录。 PostgreSQL 14 安装成功。 PostgreSQL 14 安装成功。 `psql` 安装成功。

```
SELECT version();
```

RPM 安装 `rpm` 安装成功: `rpm -qa | grep postgresql`

2. 安装 PostGIS 成功。

```
SELECT PostGIS_Full_Version();
```

安装 PostgreSQL, Proj4 和 GEOS 成功。

1. 安装 `postgis_config.hh` 成功。 `POSTGIS_PGSQL_VERSION`, `POSTGIS_PROJ_VERSION` and `POSTGIS_GEOS_VERSION` 安装成功。

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](#).

PostgreSQL configuration can be set at the server level without touching `postgresql.conf` or `postgresql.a` 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 of "partition" forces the planner to only analyze tables for constraint consideration if they are in an inherited hierarchy and avoids paying the planner penalty otherwise.

`shared_buffers`

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

`max_worker_processes` controls the maximum number of background processes available to the server and therefore also bounds how many workers can participate in 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

Parallel query support in modern PostgreSQL and PostGIS uses this setting to decide how many workers a plan may request. 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.

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. You can also 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/MAJOR/CLUSTER/environment` where the placeholders refer to the PostgreSQL major version and cluster name.

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 安装与配置

3.3.1 Spatially enable database using EXTENSION

If you 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-3.6/

# 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 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.

Use the **PostGIS_Extensions_Upgrade** function to upgrade to the latest version you have installed.

```
SELECT postgis_extensions_upgrade();
```

If you are upgrading a database that still runs PostGIS 2.5, issue an **ALTER EXTENSION** followed by `postgis_extensions_upgrade()` to make sure the legacy `postgis_raster` objects are folded back into the main extension.

```
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 "3.7.0alpha1";  
ALTER EXTENSION postgis_topology UPDATE TO "3.7.0alpha1";
```

If you get an error notice something like:

```
No migration path defined for b'...'b' to 3.7.0alpha1
```

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 "3.7.0alpha1" 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 "3.7.0alpha1next";  
ALTER EXTENSION postgis_topology UPDATE TO "3.7.0alpha1next";
```

**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 without extensions

Extensions have been the supported installation method for many releases. If you still rely on an unpackaged installation, perform a **HARD UPGRADE** into a database created with extensions and migrate your data there.

3.4.1.3 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 (简介) 数据类型

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. The coordinate dimension is at least 2, because every geometry has at least X and Y coordinates.

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, polygonal types have dimension 2, and solid types have dimension 3. 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. For example, the boundary of a Polygon is its rings (LineStrings), the boundary of a LineString is its endpoints (Points), and the boundary of a Point is empty.

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 its 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. (Two polygons are adjacent if they share an edge. They touch if they share only points or edges on their boundaries). For more details on MultiPolygon validity, see [Section 4.4](#).

```
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))
```


4.1.2.4 MultiCurve

MULTICURVE 数据类型, 用于存储由一个或多个曲线组成的几何体。

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

4.1.2.5 MultiSurface

MULTISURFACE 数据类型, (目前) 用于存储由一个或多个曲面组成的几何体。

```
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 都是用于存储和传输几何数据的标准格式。

WKT(Well-Known Text) 是一种文本格式。WKT SRS 是指 WKT 中的空间参考系统。

- 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)。

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

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

- POINT(0 0)

WKB: 010100000000000000000000F03F000000000000F03

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

WKB: 010200000002000000000000000000004000000000000000400000000000022400000000000000

Input and output of WKB is provided by the functions **ST_AsBinary** and **ST_GeomFromWKB**:

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

OGC 010100000000000000000000F03F000000000000F03F:

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

4.2 Geometry Data Type

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

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



Note

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

OpenGIS 010100000000000000000000F03F000000000000F03F (SRID) 010100000000000000000000F03F000000000000F03F. 010100000000000000000000F03F000000000000F03F SRID 010100000000000000000000F03F000000000000F03F.

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

4.2.1 OpenGIS WKB 与 WKT

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

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)))

PostGIS 3.7.0alpha1 简体中文版

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

PostGIS 3.7.0alpha1 简体中文版

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

4.3 PostGIS 地理数据类型

地理数据类型 (如“点”、“线”、“面”、“体”) 与几何数据类型类似, 但它们与地理坐标系 (如 WGS 1984) 相关联。地理数据类型 (如“点”) 与几何数据类型 (如“点”) 类似。

PostGIS 3.7.0alpha1 简体中文版。地理数据类型 (如“点”) 与几何数据类型 (如“点”) 类似。地理数据类型 (如“点”) 与几何数据类型 (如“点”) 类似。

PostGIS 3.7.0alpha1 简体中文版。地理数据类型 (如“点”) 与几何数据类型 (如“点”) 类似。地理数据类型 (如“点”) 与几何数据类型 (如“点”) 类似。地理数据类型 (如“点”) 与几何数据类型 (如“点”) 类似。

地理数据类型 (如“点”) 与几何数据类型 (如“点”) 类似。地理数据类型 (如“点”) 与几何数据类型 (如“点”) 类似。地理数据类型 (如“点”) 与几何数据类型 (如“点”) 类似。

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. 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.

4.3.2 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
-----
0101000020AD100000000000000000C05EC00000000000004140
```

```
-- NAD27 lon/lat
SELECT 'SRID=4267;POINT(-123 34)::geography;
       geography
-----
0101000020AB100000000000000000C05EC00000000000004140
```

NAD83 UTM zone meters fails because 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。

```
-- 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 性能优化与配置

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. 数据库引擎配置：确保数据库引擎（如 PostgreSQL）配置正确，特别是内存和缓存设置。

- 调整内存配置：增加共享内存（shared_buffers）和写入缓冲区（effective_cache_size）。
- 索引优化：确保查询的列上有适当的索引，并定期维护索引。
- 查询优化：避免复杂的子查询，使用 EXPLAIN 分析查询计划。

2. 硬件配置：使用 SSD 存储，确保足够的内存，并考虑使用 RAID 配置。

4.3.4 常见问题 FAQ

1. 为什么我的查询速度很慢？

By default, all distance and area calculations are done on the spheroid. You should find that the results of calculations in local areas match up well with local planar results in good local projections. Over larger areas, the spheroidal calculations will be more accurate than any calculation done on a projected plane. The difference is that the spheroid takes into account the curvature and non-spherical shape of the Earth, whereas planar calculations assume a flat surface. 检查你的查询是否使用了正确的投影，并考虑使用平面投影进行局部计算。

2. 如何优化我的查询？

使用 EXPLAIN 分析查询计划，了解数据库引擎如何执行查询。优化索引和查询语句。

3. 为什么我的距离计算结果不准确？

PostGIS 默认使用球面模型（great circle arc）进行距离计算。如果你需要平面投影的结果，可以使用 ST_Distance_Sphere 函数。

4. 为什么我的查询返回了错误的结果？

检查你的查询是否使用了正确的投影，并考虑使用平面投影进行局部计算。确保你的数据是最新的，并且没有重复的记录。

4.4 Geometry Validation

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

4.4.1 Simple Geometry

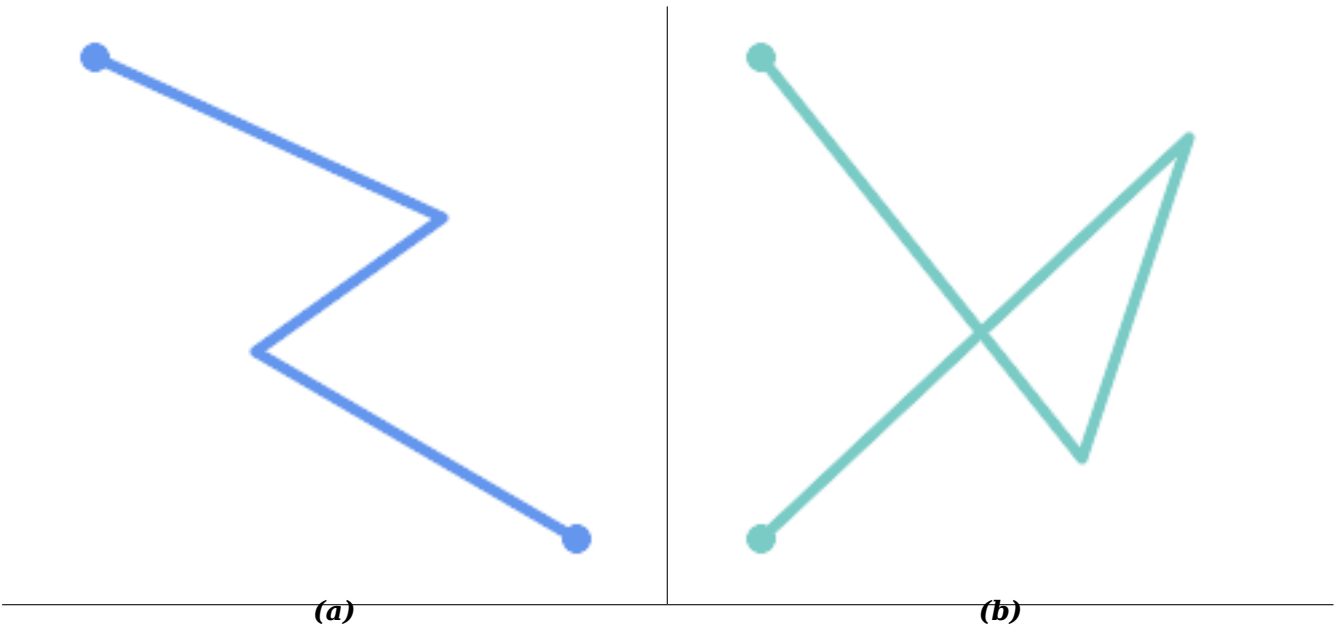
A *simple* geometry is one that has no anomalous geometric points, such as self intersection or self tangency.

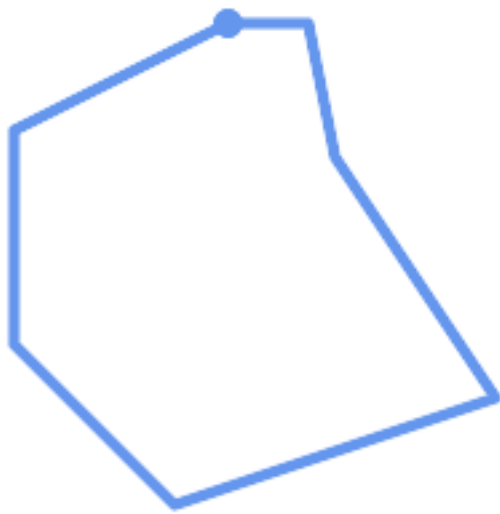
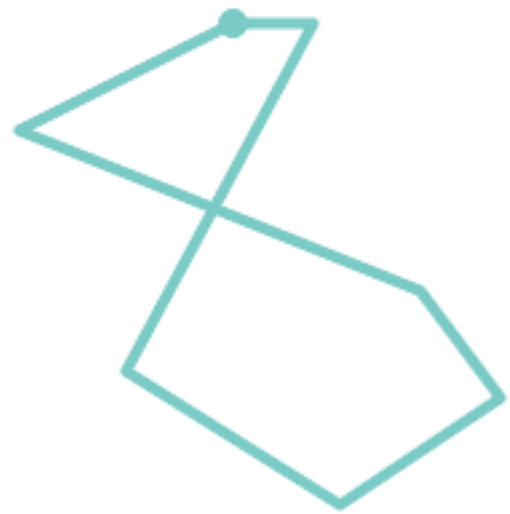
POINT 0 0000000000000000 000000.

MULTIPOINT 000000 (POINT) 0000000 (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.

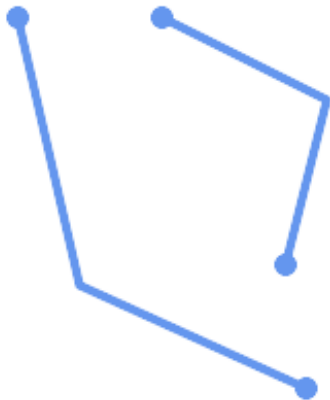
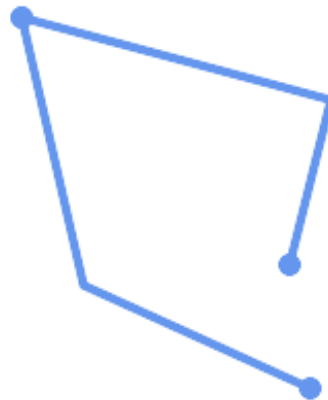
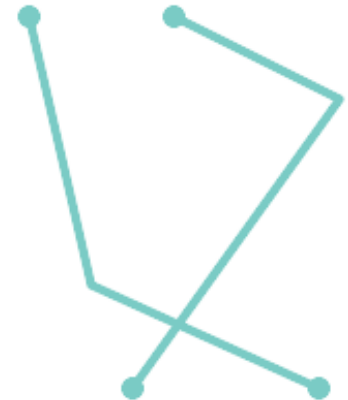
(a) and **(c)** are simple LINESTRINGS. **(b)** and **(d)** are not simple. **(c)** is a closed Linear Ring.



**(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

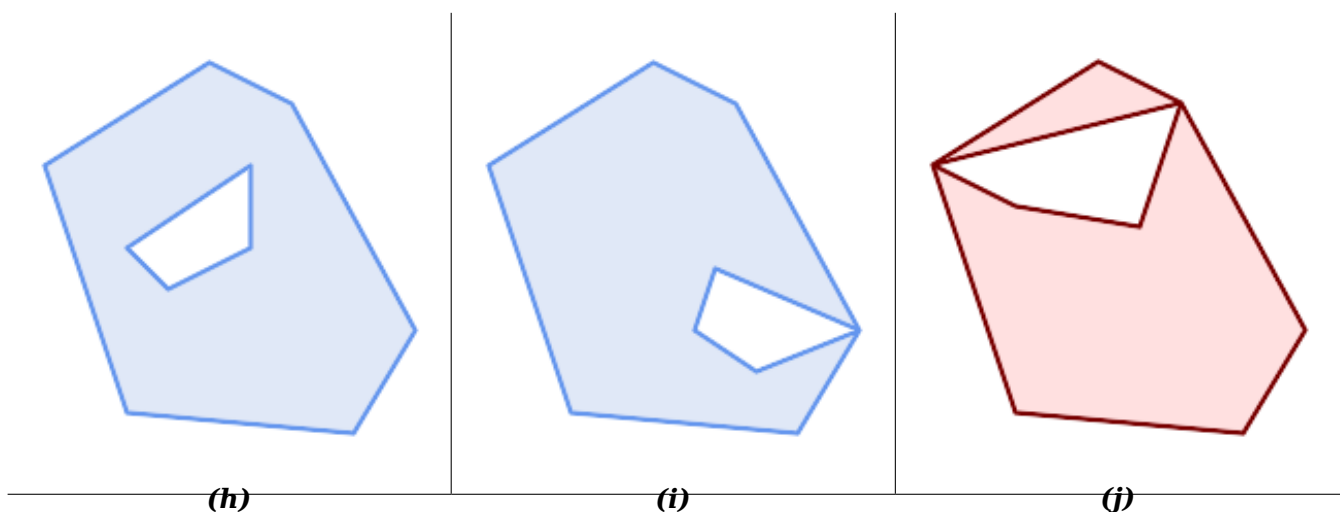
Geometry validity primarily applies to 2-dimensional geometries (POLYGONs and MULTIPOLYGONs) . Validity is defined by rules that allow polygonal geometry to model planar areas unambiguously.

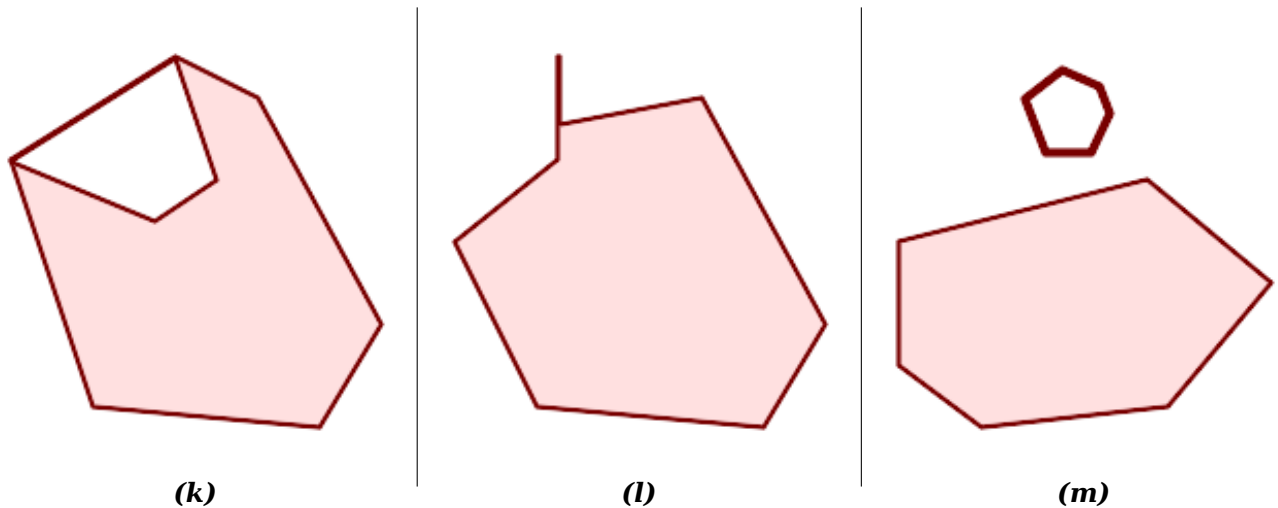
A POLYGON is *valid* if:

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)

Consecutive repeated points in a polygon boundary ring are treated as redundant vertices, and do not make the polygon invalid. Repeated points that cause the boundary to self-intersect, self-touch, or collapse still violate the validity rules.

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

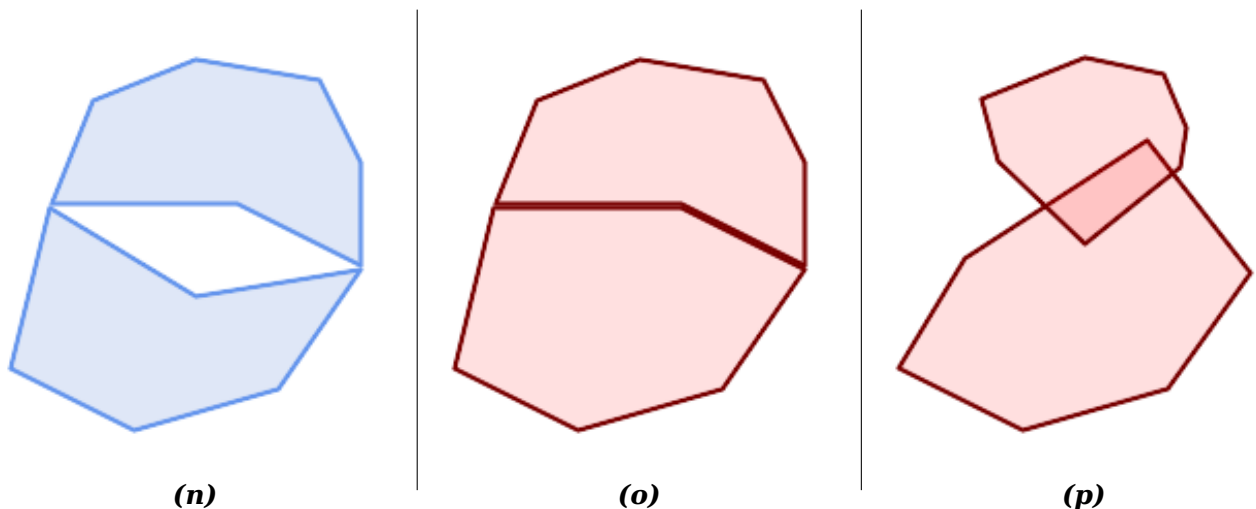




A MULTIPOLYGON is *valid* if:

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

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



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

For linear geometry the only validity rule is that 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 141.56976744186048)

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

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

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

4.5 SPATIAL_REF_SYS

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

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

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

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

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

4.5.1 SPATIAL_REF_SYS Table

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

SPATIAL_REF_SYS 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 权威名称。如果为“EPSG”，则为 EPSG 代码。
AUTH_NAME 列。

auth_srid 权威 ID。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。WKT SRS 描述:

```
PROJCS["NAD83 / UTM Zone 10N",
  GEOGCS["NAD83",
    DATUM["North_American_Datum_1983",
      SPHEROID["GRS 1980",6378137,298.257222101]
    ],
```



```
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 几何数据类型

4.6.1 创建几何表

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. 修饰符 Z, M 或 ZM 指定三维或四维几何。例如，geometry('LINESTRINGM' 3005) 指定三维 LineString 在 SRID 3005 的坐标系中。geometry('POINTZM' 3005) 指定四维点在 SRID 3005 的坐标系中。
- 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);
```


If the geometry type is known to be a 2D Polygon, specify it in the cast.

```
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;
```

For a derivative table created by a bulk insert, create the table first.

```
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 a 2D spatial index on the new table.

```
CREATE INDEX idx_myschema_myspecialpois_geom_gist
ON myschema.my_special_pois USING gist(geom);
```

If the points are 3D or measured, an n-D index may be more appropriate.

```
CREATE INDEX my_special_pois_geom_gist_nd
ON my_special_pois USING gist(geom gist_geometry_ops_nd);
```

Manually register the new table's geometry column in `geometry_columns`. This also changes the underlying table structure to make the column `typmod`-based.

```
SELECT Populate_Geometry_Columns('myschema.my_special_pois'::regclass);
```

To retain constraint-based definition behavior, such as for inherited tables where all children do not have the same type and SRID, set the optional `use_typmod` argument to false.

```
SELECT Populate_Geometry_Columns('myschema.my_special_pois'::regclass, false);
```

Although the old constraint-based method is still supported, direct pass-through view columns register correctly in `geometry_columns` for both `typmod`-based and constraint-based source columns. In this example we define a column using `typmod` and another using constraints.

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

If we run `\d pois_ny` in `psql`, we observe they are defined differently -- one is `typmod`, one is constraint.

```
\d pois_ny
```

Column	Type	Table "public.pois_ny"	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`

```
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';
```

Both the typmod based geom view column and the direct constraint based view column register correctly.

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

If the view applies a spatial function to the geometry, such as ST_Transform, you still need to cast the transformed column explicitly so the view can expose the correct type and SRID:

```
DROP VIEW vw_pois_ny_parks;
CREATE VIEW vw_pois_ny_parks AS
SELECT gid, poi_name, cat,
geom,
ST_Transform(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 (GIS) 数据源

PostGIS 支持多种 GIS 数据源。目前，PostGIS 支持 shapefile 和 PostGIS/PostgreSQL 数据库。PostGIS 还支持其他 GIS 数据源，如 Oracle Spatial、Microsoft SQL Server 等。

4.7.1 SQL 脚本格式

PostGIS 支持使用 (formatted) SQL 脚本格式。Oracle SQL 脚本格式, SQL “INSERT” 脚本格式 (piping) 等脚本格式。

以下脚本 (roads.sql) 用于创建 roads 表:

```
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 脚本格式 PostgreSQL 脚本格式。

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

4.7.2 shp2pgsql: ESRI shapefile 脚本

shp2pgsql 脚本 ESRI shapefile 文件, 将其加载到 PostgreSQL/PostGIS 数据库中。脚本格式 (command line) 脚本格式。

The input must be an unpacked shapefile path. The loader does not read .zip archives directly; unpack the .shp, .shx, and .dbf files before loading.

shp2pgsql 脚本, 用于 PostgreSQL 数据库。shp2pgsql-gui 脚本 pgAdmin III 脚本。

c|a|d|p -- 脚本格式:

- c** 脚本格式 shapefile 脚本格式。脚本格式。
- a** 脚本格式 shapefile 脚本格式。脚本格式。
- d** 脚本格式 (drop) 脚本 shapefile 脚本格式。
- p** 脚本格式 SQL 脚本格式, 脚本格式。脚本格式。

-? 脚本格式。

-D 脚本格式 PostgreSQL “脚本 (dump)” 脚本格式。脚本格式 -a, -c 脚本格式 脚本格式。脚本格式 “脚本” SQL 脚本格式。脚本格式。

-f <fid_column>, --feature-id-column <fid_column> Specify the name of the feature id column. The default is gid. The value must be a simple PostgreSQL identifier and cannot use a PostgreSQL system column name.


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

UNIX 管道 (pipe) 将 shp2pgsql 的输出直接传递给 psql:

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

4.8 地理函数

SQL 将 shapefile 加载到 PostgreSQL/PostGIS 数据库中。SQL 将查询结果输出到文本文件。

4.8.1 SQL 查询

以下 SQL 查询 (select) 从 roads 表中选择所有记录:

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

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

(6 rows)

以下 SQL 查询将 roads 表中的几何数据转换为文本格式。结果将显示为文本格式。

ST_Intersects This function tells whether two geometries share any space.

= 将两个几何体进行比较。例如，'POLYGON((0 0,1 1,1 0,0 0))' 与 'POLYGON((0 0,1 1,1 0,0 0))' 是否相交。

以下 SQL 查询将 roads 表中的几何数据转换为文本格式。结果将显示为文本格式。

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

以下 SQL 查询将 roads 表中的几何数据转换为文本格式。结果将显示为文本格式。

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

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

“map frame”) 和 “frame-based”) 的差别。

“&&” 运算符, 使用 BOX3D 对象。因此, 使用 “&&” 运算符, 使用 BOX3D 对象。

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

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

使用 SRID 312 的 BOX3D 对象。

4.8.2 使用 pgsql2shp

pgsql2shp 是一个命令行工具, 用于将 shapefile 转换为 PostgreSQL 数据库。

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

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

使用以下选项:

- f <filename> 指定要加载的 shapefile 文件名。
- h <host> 指定数据库主机名。
- p <port> 指定数据库端口号。
- P <password> 指定数据库密码。
- u <user> 指定数据库用户名。
- g <geometry column> 指定 shapefile 中的几何列名。
- b 指定几何列的数据类型。默认是 GEOMETRY。如果指定了数据类型, 则使用 (cast) 指定数据类型 (非) 指定数据类型。
- r 指定 (raw) 数据类型。gid 指定几何列的 ID 列名。
- m filename 指定 10 个字符的文件名 (remap) 文件。VERYLONGSYMBOL SHORTONE ANOTHERVERYLONGSYMBOL SHORTER 指定文件名。

4.9 索引

PostgreSQL 支持多种索引类型, 包括 B-Tree, R-Tree, GiST 等。PostgreSQL 还支持全文索引 (Full-Text Index)。

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“多键”索引，它支持对多键数据进行索引。PostGIS 使用 GiST 索引 GIS 数据。与 B-Tree 索引相比，GiST 索引在多维数据上具有更好的性能。
- **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 querying with spatial predicates see Section 5.2.

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

4.9.1 GiST 索引

GiST 是一种“多键”索引，它支持对多键数据进行索引。GIS 数据，如 B-Tree 索引，在多维数据上具有更好的性能。与 B-Tree 索引相比，GiST 索引在多维数据上具有更好的性能。

GIS 数据，如 B-Tree 索引，在多维数据上具有更好的性能。与 B-Tree 索引相比，GiST 索引在多维数据上具有更好的性能。

“多键”索引 GiST 索引在多维数据上具有更好的性能。

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

对于 2D 数据，PostGIS 2.0 版本中，n 维数据，如 B-Tree 索引，在多维数据上具有更好的性能。

```
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 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 provided by PostgreSQL. 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 used to speed up queries that involve spatial data. They are used to index the geometry column of a table. The index is created by grouping the data into blocks, and then indexing the blocks. The index is used to find the blocks that contain the data being searched for. The index is then used to find the data in the blocks.

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;
```

” ” GiST

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

2D 2D. PostGIS 2.0 n

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

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

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

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

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

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

" 2D " 2D GiST 索引类:

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

PostGIS 2D 索引类。PostGIS 2.0 引入的 `n` 索引类, 索引类:

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 索引类, 索引类。索引类, 索引类。

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


Chapter 5

Spatial Queries

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

5.1 Determining Spatial Relationships

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

5.1.1 Dimensionally Extended 9-Intersection Model

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

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

Boundary

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

Interior

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

Exterior

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

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

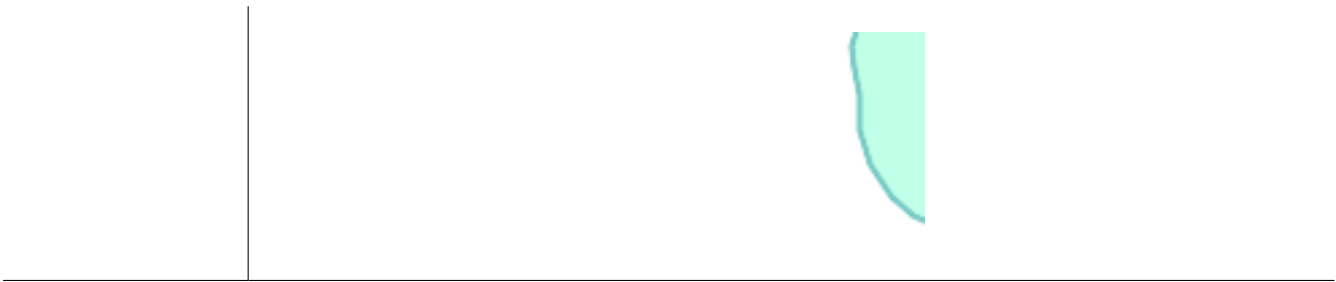
For a geometry g the *Interior*, *Boundary*, and *Exterior* are denoted using the notation $I(g)$, $B(g)$, and $E(g)$. Also, $dim(s)$ denotes the dimension of a set s with the domain of $\{0, 1, 2, F\}$:

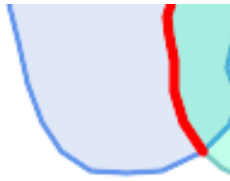
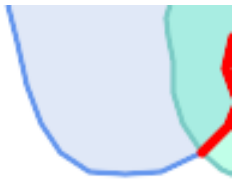
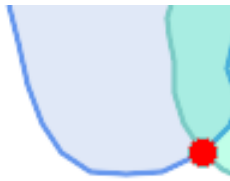
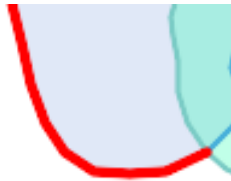
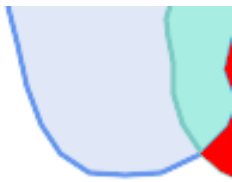
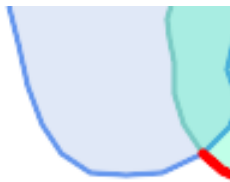
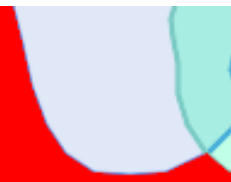
- 0 => point
- 1 => line
- 2 => area
- F => empty set

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

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

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



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

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

For more information, refer to:

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

5.1.2 Named Spatial Relationships

To make it easy to determine common spatial relationships, the OGC SFS defines a set of *named spatial relationship predicates*. PostGIS provides these as the functions **ST_Contains**, **ST_Crosses**, **ST_Disjoint**, **ST_Equals**, **ST_Intersects**, **ST_Overlaps**, **ST_Touches**, **ST_Within**. It also defines the non-standard relationship predicates **ST_Covers**, **ST_CoveredBy**, and **ST_ContainsProperly**.

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

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

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

5.1.3 General Spatial Relationships

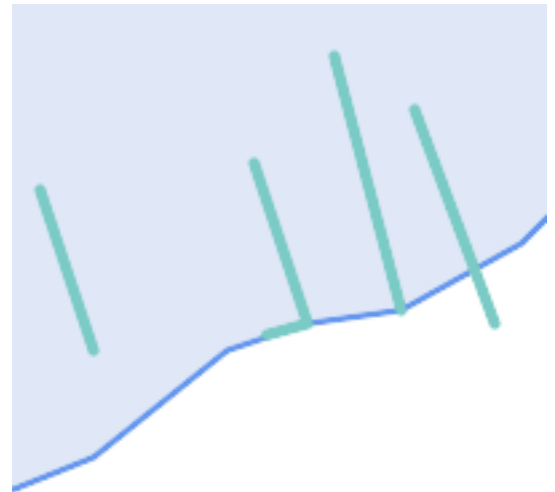
In some cases the named spatial relationships are insufficient to provide a desired spatial filter condition.



For example, consider a linear dataset representing a road network. It may be required to identify all road segments that cross each other, not at a point, but in a line (perhaps to validate a business rule such as ensuring that no two different roads share a common segment). 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))' );
```

```
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	1539.476
VANCOUVER	1450.331
LANGLEY DISTRICT	833.793
BURNABY	773.769
PRINCE GEORGE	694.376
...	

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

TOAST

6.1 TOAST 简介

6.1.1 背景

Current PostgreSQL versions suffer from a query optimizer weakness regarding TOAST tables. TOAST tables are a kind of "extension room" used to store large (in the sense of data size) values that do not fit into normal data pages (like long texts, images or complex geometries with lots of vertices), see [the PostgreSQL Documentation for TOAST](#) for more information).

在 PostgreSQL 中，TOAST 表用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。

在 PostgreSQL 中，TOAST 表用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。

在 PostgreSQL 中，TOAST 表用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。

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

6.1.2 使用

PostgreSQL 使用 TOAST 表来存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。

在 PostgreSQL 中，TOAST 表用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。

在 PostgreSQL 中，TOAST 表用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。TOAST 表是一种“扩展空间”，用于存储那些无法放入普通数据页（如长文本、图像或具有大量顶点的复杂几何体）的大值。

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


Chapter 7

PostGIS Reference

PostGIS is a spatial database system that provides the ability to store and retrieve geographic data. It is an extension of PostgreSQL, and it provides a rich set of functions for working with spatial data.



Note
PostGIS is a spatial database system that provides the ability to store and retrieve geographic data. It is an extension of PostgreSQL, and it provides a rich set of functions for working with spatial data.

7.1 PostgreSQL PostGIS Geometry/Geography/Box

7.1.1 box2d

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

box3d is a data type in PostgreSQL that represents a 3-dimensional bounding box. It is an extension of the box2d type, and it provides a rich set of functions for working with 3D spatial data.

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 版本中引入。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 版本中引入。

图例

geography 数据类型在 PostgreSQL 9.5 版本中引入。

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

图例

图例

geometry_dump	geometry_dump
box	geometry
box2d	geometry
box3d	geometry
bytea	geometry
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 地理信息系统的几何类型。

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

地理信息

地理信息系统的几何类型。

geometry	geometry
geometry	geometry

注意

Section 4.3, Section 4.3

7.2 几何类型

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_typmod=true);
text **AddGeometryColumn**(varchar schema_name, varchar table_name, varchar column_name, integer srid, varchar type, integer dimension, boolean use_typmod=true);
text **AddGeometryColumn**(varchar catalog_name, varchar schema_name, varchar table_name, varchar column_name, integer srid, varchar type, integer dimension, boolean use_typmod=true);

注意

schema_name 默认为当前数据库的 schema_name。srid 默认为 4326。type 默认为 'POLYGON' 或 'MULTILINESTRING'。search_path 默认为 SRID, 数据类型, 几何列名。



Note

注意: 2.0.0 版本 geometry_columns 表使用 typmod 值。PostgreSQL 使用 WGS84 POINT 类型。ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326); 2.0.0 版本。use_typmod 默认为 true。



Note

注意: 2.0.0 版本。geometry_columns 表使用 typmod 值。PostgreSQL 使用 WGS84 POINT 类型。ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326); 2.0.0 版本。use_typmod 默认为 true。Section 4.6.3 详细描述了 typmod 值。


```
"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)
```

The 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 — [DropGeometryColumn](#).

Synopsis

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

[DropGeometryColumn](#). schema_name [geometry_columns](#) [f_table_schema](#).

- ✔ 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
PostGIS 2.0.0. [DropGeometryColumn](#). [geometry_columns](#) [ALTER TABLE](#).

❏

```
SELECT DropGeometryColumn ('my_schema','my_spatial_table','geom');
```

```
my_schema.my_spatial_table.geom effectively removed.
```

The above is also equivalent to the standard **ALTER TABLE** command. Both deregister the column from `geometry_columns`.

```
ALTER TABLE my_schema.my_spatial_table DROP column geom;
```

❏

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

7.2.3 DropGeometryTable

`DropGeometryTable` — 删除 `geometry_columns` 表。

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` 表。: 当前数据库。: 当前数据库 (schema-aware) `pgSQL` `current_schema()` 函数。



Note

PostGIS 2.0.0 版本。删除 `geometry_columns` 表。删除 `geometry_columns` 表时，`DROP TABLE` 命令将删除该表。

❏

```
SELECT DropGeometryTable ('my_schema','my_spatial_table');
```

```
my_schema.my_spatial_table dropped.
```

The above is now equivalent to the standard **DROP TABLE** command.

```
DROP TABLE my_schema.my_spatial_table;
```

❏

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

oid 数据类型, 数据类型 SRID, 数据类型, 数据类型
数据类型. 数据类型 geometry_columns 数据类型
数据类型. 数据类型, 数据类型.

oid 数据类型 oid 数据类型, 数据类型 geometry_columns 数据类型
数据类型 SRID, 数据类型, 数据类型, 数据类型.

数据类型, 数据类型
数据类型 geometry_columns 数据类型 (truncate) 数据类型
数据类型. 数据类型 geometry_columns 数据类型
数据类型. 数据类型 geometry_columns 数据类型
数据类型.

1.4.0 数据类型.

数据类型: 2.0.0 数据类型. 数据类型. 数据类型 use_typmod
数据类型, 数据类型.

数据类型: 2.0.0 数据类型. 数据类型 use_typmod
数据类型.

数据类型

This example populates geometry column metadata using typmod behavior. For this to work, there must be existing data.

```
CREATE TABLE public.myspatial_table(gid serial, geom geometry);
INSERT INTO myspatial_table(geom) VALUES(ST_GeomFromText('LINESTRING(1 2, 3 4)',4326) );
SELECT Populate_Geometry_Columns('public.myspatial_table'::regclass);
```

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 example changes the geometry columns to use constraints if they are not typmod or already constrained. For this to work, there must be existing 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);
```

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);
```

更新

`UpdateGeometrySRID`, `geometry_columns` 表中的 `srid` 列将被更新为 `SRID` 参数值。注意：在 schema-aware postgres 安装中，`current_schema()` 可用于



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



This method supports Circular Strings and Curves.

更新

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)
\.
```

更新 `SRID` 列中的 `SRID` 为 4326:

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

更新 DDL 中的 `SRID`:

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

更新 `SRID` 列中的 `SRID` 为 3857 (未知) 使用 `ST_Transform` 函数:

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

更新

UpdateRasterSRID, **ST_SetSRID**, **ST_Transform**

7.3 几何集合 (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);
```

几何集合

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 版本引入。使用 **ST_MakeLine** 函数可以创建由点组成的线。使用 **ST_MakeLine** 函数可以创建由点组成的线。



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



This method supports Circular Strings and Curves.

几何集合: XLink

Collect 2D points.

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

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

```
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)'));
```

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

例: 创建多点点集

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;
```

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

例: 创建多线集

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);

返回

返回类型为 geometry。

Use **ST_MakeLine** to create lines from Point or LineString inputs.



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

返回

返回类型为 geometry。

```
SELECT ST_AsEWKT(ST_LineFromMultiPoint('MULTIPOINT(1 2 3, 4 5 6, 7 8 9)'));
```

```
LINESTRING(1 2 3,4 5 6,7 8 9)
```

返回

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。返回类型为 geometry。

Synopsis

geometry **ST_MakeEnvelope**(float xmin, float ymin, float xmax, float ymax, integer srid=unknown);

返回

返回类型为 geometry。SRID 参数为可选，默认为 4326。如果 SRID 不为 4326，则返回的几何体将使用指定的 SRID。

1.5 版本之前，返回类型为 geometry。

参数: 2.0 版本 SRID 参数 (envelope) 为可选参数。

返回: 返回类型为 geometry。

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

```
POLYGON((10 10, 10 11, 11 11, 11 10, 10 10))
```



ST_MakePoint, ST_MakePoint, ST_Point, ST_SRID

7.3.4 ST_MakeLine

[illegible]

Synopsis

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



Creates a LineString containing the points of Point, MultiPoint, or LineString geometries. Other geometry types cause an error.

Variant 1: accepts two input geometries

Variant 2: accepts an array of geometries

Variant 3: aggregate function accepting a rowset of geometries. To ensure the order of the input geometries use `ORDER BY` in the function call, or a subquery with an `ORDER BY` clause.

Repeated nodes at the beginning of input LineStrings are collapsed to a single point. Repeated points in Point and MultiPoint inputs are not collapsed. Components of MultiLineString are handled in the order they appear in the collection. **ST_RemoveRepeatedPoints** can be used to collapse repeated points from the output LineString.

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

Availability: 3.7.0 - Support for MultiLineString input elements was introduced

2.0.0 □□□□□□□□□□□□□□□□□□□□□□□□□□.

[illegible]

1.4.0 ST MakeLine ST MakeLine

☐ ☐ : ☐☐☐☐☐☐☐

Create a line composed of two points.

```
SELECT ST_AsText(ST_MakeLine(ST_Point(1,2), ST_Point(3,4)));
```

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

3D 2 BOX3D.

```
SELECT ST AsEWKT(ST MakeLine(ST MakePoint(1,2,3), ST MakePoint(3,4,5)));
```

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

[illegible]

```
SELECT ST AsText(ST MakeLine('LINESTRING(0 0, 1 1)', 'LINESTRING(2 2, 3 3)'));
```

```
LINESTRING(0 0,1 1,2 2,3 3)
```

名称: ST_MakeLine

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)]));
```

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

名称: ST_MakeLine

使用 GPS 轨迹点，按时间顺序生成 GPS 轨迹线。GPS 轨迹点按时间顺序排列。

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;
```

When ordering within the aggregate is not possible, 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;
```

名称:

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.

☒☒

This example creates a point with an unknown SRID.

```
SELECT ST_MakePoint(-71.1043443253471, 42.3150676015829);
```

This example creates a point in the WGS 84 geodetic CRS.

```
SELECT ST_SetSRID(ST_MakePoint(-71.1043443253471, 42.3150676015829),4326);
```

This example creates a 3D point with altitude.

```
SELECT ST_MakePoint(1, 2,1.5);
```

This example returns the Z coordinate of a point.

```
SELECT ST_Z(ST_MakePoint(1, 2,1.5));
```

```
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 经纬度坐标。

Synopsis

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

国国

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

Use **ST_SetSRID** to specify a SRID for the created point.



Note

For geodetic coordinates, X is longitude and Y is latitude



Note

The functions **ST_PointM**, and **ST_PointZM** can be used to create points with an M value and a given SRID.

国国



Note

ST_AsEWKT is used for text output because **ST_AsText** does not support M values.

Create point with unknown SRID.

```
SELECT ST_AsEWKT(ST_MakePointM(-71.1043443253471, 42.3150676015829, 10));
```

```
POINTM(-71.1043443253471 42.3150676015829 10)
```

x, y 国国国国国国国国国国国国国国国国.

```
SELECT ST_AsEWKT(ST_SetSRID(ST_MakePointM(-71.104, 42.315, 10), 4326));
```

```
SRID=4326;POINTM(-71.104 42.315 10)
```

Get measure of created point.

```
SELECT ST_M(ST_MakePointM(-71.104, 42.315, 10));
```

```
10
```

国国

ST_MakePoint, **ST_SetSRID**, **ST_PointM**, **ST_PointZM**

7.3.7 ST_MakePolygon

ST_MakePolygon — Creates a Polygon or CurvePolygon from a shell and optional list of holes.

Synopsis

```
geometry ST_MakePolygon(geometry linestring);
```

```
geometry ST_MakePolygon(geometry outerlinestring, geometry[] interiorlinestrings);
```



Creates a Polygon or CurvePolygon formed by the given shell and optional array of holes. Input geometries must be closed LineStrings, CircularStrings, CompoundCurves, or NURBSCurves (rings). If any input ring is curved, the result is a CurvePolygon.

Variant 1: Accepts one shell ring.

Variant 2: Accepts a shell ring and an array of inner (hole) rings. A geometry array can be constructed using the PostgreSQL array `agg()`, `ARRAY[]` or `ARRAY()` constructs.



Note

ST Dump ST_LineMerge



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

Enhanced: 3.7.0 - Support for curved input rings was introduced.

[illegible][illegible]

```
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;
```

[illegible]

```
SELECT ST_AsEWKT(ST_MakePolygon('LINESTRING(75.15 29.53 1,77 29 1,77.6 29.5 1, 75.15 29.53 1)'));
```

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)'));
```

POLYGONM((75.15 29.53 1,77 29 1,77.6 29.5 2,75.15 29.53 2))

Create a CurvePolygon from a CircularString shell.

```
SELECT ST_AsText(ST_MakePolygon('CIRCULARSTRING(0 0,1 1,2 0,1 -1,0 0)'));
```

```
CURVEPOLYGON(CIRCULARSTRING(0 0,1 1,2 0,1 -1,0 0))
```

注意: 使用 ST_MakePolygon 时，如果输入的多边形包含孔，则必须使用 ST_MakePolygon 的第二个参数来指定孔。

以下 SQL 语句演示了如何使用 ST_MakePolygon 来创建带有孔的多边形。

```
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 值在 ST_MakePolygon 中会被忽略，因此使用 CASE 语句来避免 NULL 值。

```
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;
```

￼￼

[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);

￼￼

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

￼￼: ￼￼

```
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);
```

￼￼: ￼￼￼

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;
```

☒☒

[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);

☒☒

☒☒☒☒☒☒☒☒ 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_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)
```

☒☒

[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);

☒☒

☒☒☒☒☒☒☒☒ 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_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);

返回

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

此函数与 [ST_LineMerge](#) 和 [ST_Dump](#) 一起使用。

- ✓ 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.

返回

Create a 2D polygon.

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

```
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));
```

```
SRID=4326;POLYGON((75 29 1, 77 29 2, 77 29 3, 75 29 1))
```

返回

[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);

ST_TileEnvelope

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**.

语法: 2.0.0 `ST_TileEnvelope(zoom, xtile, ytile, SRID, bounds)`

2.1.0 `ST_TileEnvelope(zoom, xtile, ytile, SRID, bounds, margin)`

参数: `zoom`: 瓦片的缩放级别

```
SELECT ST_AsText(ST_TileEnvelope(2, 1, 1));
```

```
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)));
```

```
POLYGON((-135 45,-135 67.5,-90 67.5,-90 45,-135 45))
```

ST_MakeEnvelope

ST_MakeEnvelope

7.3.14 ST_HexagonGrid

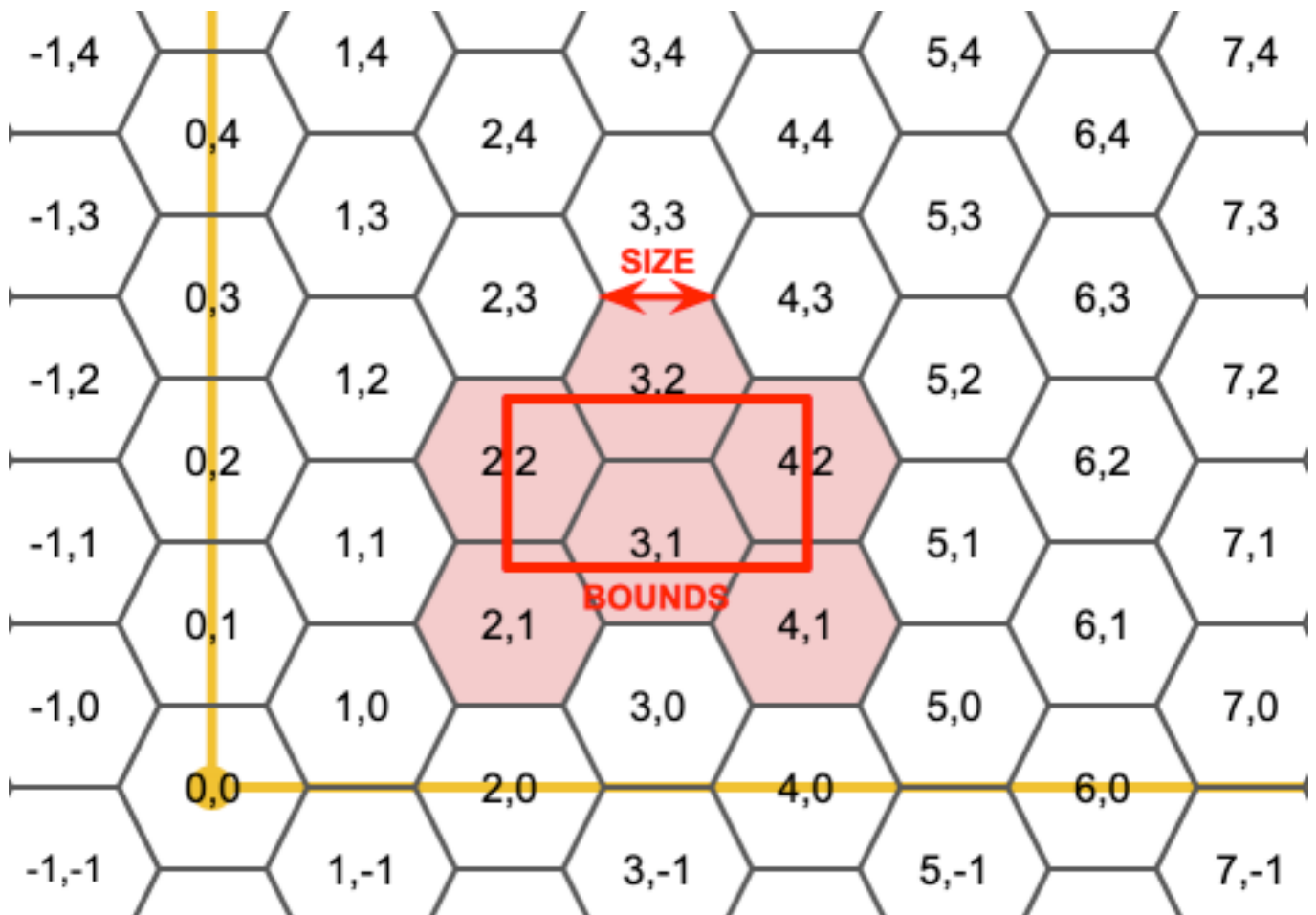
ST_HexagonGrid — Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.

Synopsis

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

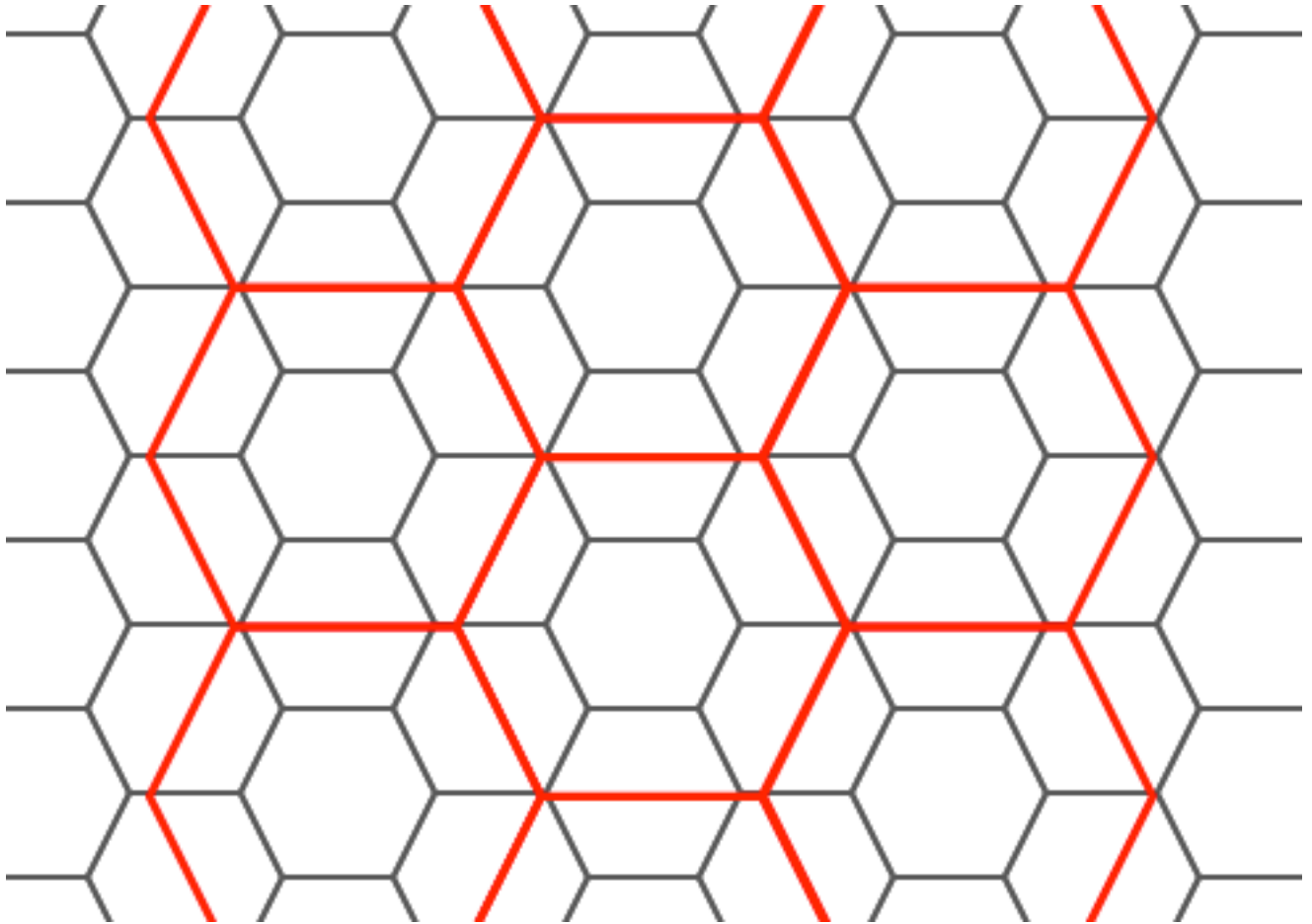
￼￼

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 创建六边形网格。

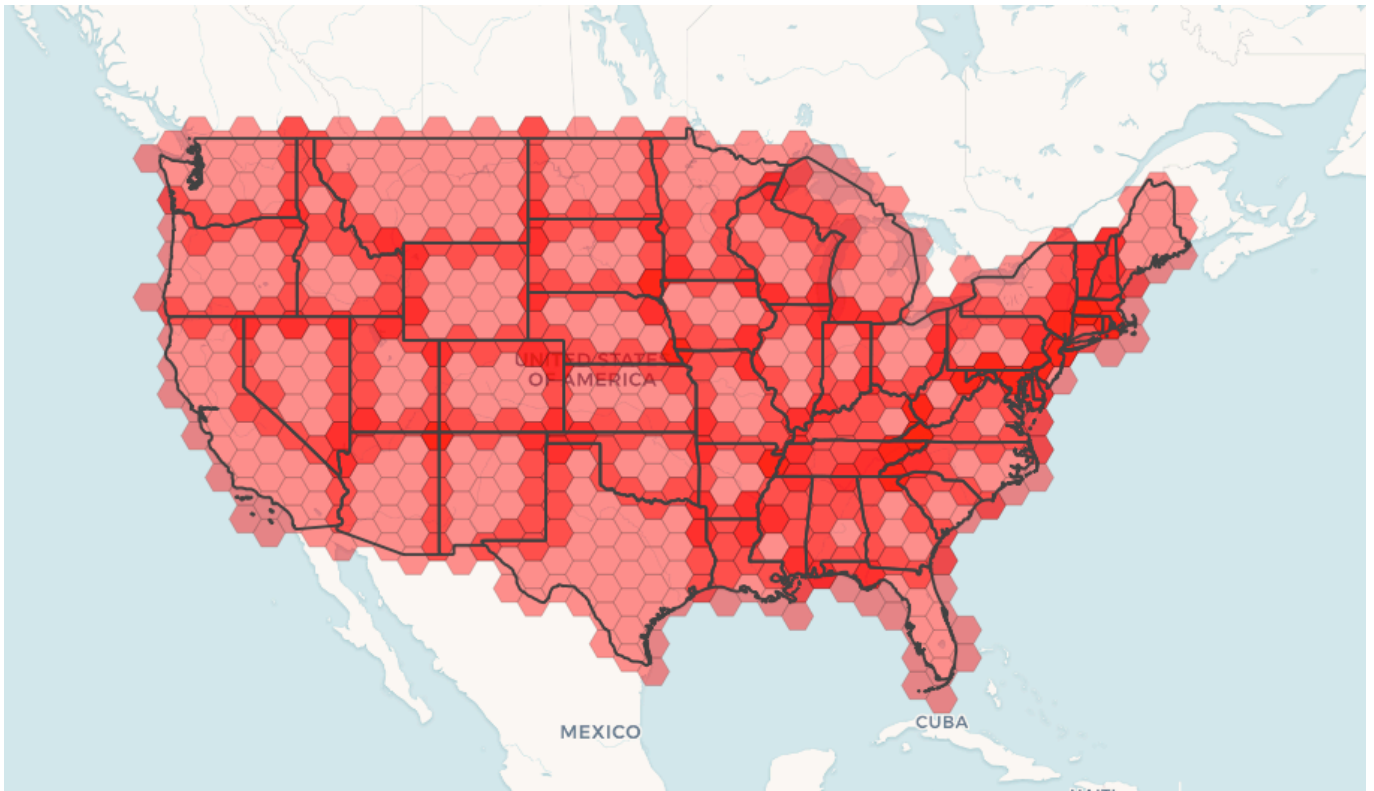
图例: 六边形网格

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: 多边形边界上的六边形网格

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: 点数据和网格

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: 点数据和网格

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 'Yo' 的几何图形

```
SELECT ST_AsText(ST_Letters('Yo'));
```



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](#)

7.3.19 ST_MakeNurbsCurve

ST_MakeNurbsCurve — Creates a NURBS (Non-Uniform Rational B-Spline) curve from control points, optional weights, and optional knot vector.

Synopsis

geometry **ST_MakeNurbsCurve**(integer degree, geometry control_points, float8[] weights, float8[] knots);

例

Creates a NURBS (Non-Uniform Rational B-Spline) curve geometry from the specified parameters. NURBS curves are smooth parametric curves that provide precise control over curve shape through control points, weights, and knot values.

The degree parameter specifies the polynomial degree of the curve (must be between 1 and 10). Higher degrees produce smoother curves.

2.0.0 版本支持, 支持 TIN 表面。

- ✔ 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).

例

```
SELECT GeometryType(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
```

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))
) )');
POLYHEDRALSURFACE
```

```
SELECT GeometryType(geom) as result
FROM
  (SELECT
    ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))) AS geom
  ) AS g;
```

TIN

例

ST_GeometryType

7.4.2 ST_Boundary

ST_Boundary — 返回几何体的边界。

Synopsis

geometry **ST_Boundary**(geometry geomA);

几何

几何对象 (closure) 几何对象。 几何对象 (combinatorial boundary) 符合 OGC 规范 3.12.3.2 几何对象。 几何对象 (位相的) 几何对象, OGC 规范 3.12.2 几何对象 (primitive) 几何对象。

GEOS 几何



Note

2.0.0 几何对象 GEOMETRYCOLLECTION 几何对象。 2.0.0 几何对象 (几何对象) NULL 几何对象。



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

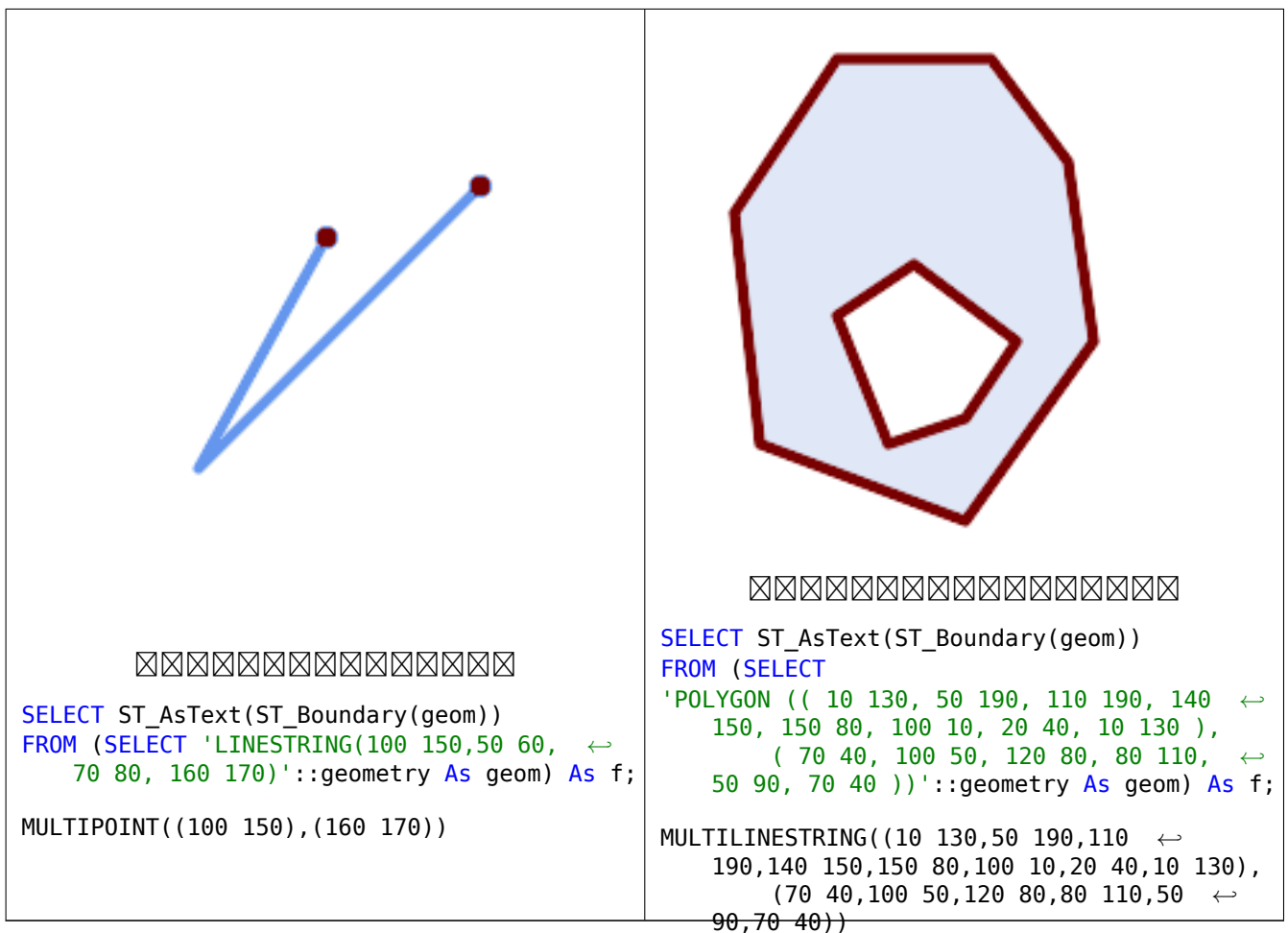


This function supports 3d and will not drop the z-index.

几何: 2.1.0 几何对象。

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

几何



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

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

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

```
LINESTRING(1 1,0 0,-1 1,1 1)
```

This example uses a 3D polygon.

```
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromEWKT('POLYGON((1 1 1,0 0 1, -1 1 1, 1 1 1))')));
```

```
LINESTRING(1 1 1,0 0 1,-1 1 1,1 1 1)
```

This example uses 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) )')));
```

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

☒☒

ST_AsText, ST_ExteriorRing, ST_MakePolygon

7.4.3 ST_BoundingDiagonal

ST_BoundingDiagonal — 返回几何体的最小外接矩形的对角线。

Synopsis

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

返回

返回几何体的最小外接矩形的对角线。如果 fits 为 true，则返回最佳拟合的对角线。如果 fits 为 false，则返回最小外接矩形的对角线。如果几何体是 2D 的，则返回最小外接矩形的对角线。如果几何体是 3D 的，则返回最小外接矩形的对角线。

fits 为 true 时 (best fit) 返回最佳拟合的对角线。如果几何体是 2D 的，则返回最小外接矩形的对角线。如果几何体是 3D 的，则返回最小外接矩形的对角线。如果几何体是 4D 的，则返回最小外接矩形的对角线。

返回的几何体具有与输入几何体相同的 SRID。



Note

如果几何体是 2D 的，则返回最小外接矩形的对角线。如果几何体是 3D 的，则返回最小外接矩形的对角线。如果几何体是 4D 的，则返回最小外接矩形的对角线。

2.2.0 版本引入。



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



This function supports M coordinates.

返回

This example returns the minimum X coordinate of a buffer around a point.

```
SELECT ST_X(ST_StartPoint(ST_BoundingDiagonal(
  ST_Buffer(ST_Point(0,0),10)
)));
```

-10

返回

ST_StartPoint, ST_EndPoint, ST_X, ST_Y, ST_Z, ST_M, &&&

7.4.4 ST_CoordDim

ST_CoordDim — 返回几何体的坐标维度。

Synopsis

integer **ST_CoordDim**(geometry geomA);

返回

ST_Geometry 数据类型。

返回 MM 值, **ST_NDims** 数据类型。

- ✓ 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)');
```

```
3
```

```
SELECT ST_CoordDim(ST_Point(1,2));
```

```
2
```

返回

ST_NDims

7.4.5 ST_Dimension

ST_Dimension — ST_Geometry 数据类型。

Synopsis

integer **ST_Dimension**(geometry g);

返回

返回的几何类型可以是 POINT, LINESTRING, POLYGON, 或者 GEOMETRYCOLLECTION。OGC 规范 s2.1.1.1 定义了 POINT 0, LINESTRING 1, POLYGON 2, 以及 GEOMETRYCOLLECTION 3。如果输入是 null，则返回 null。



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

版本: 2.0.0 支持多面体表面 (polyhedral surface) 和 TIN 表面。支持多面体表面和 TIN 表面。



Note

2.0.0 支持多面体表面和 TIN 表面。



This function supports Polyhedral surfaces.



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

返回

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

```
1
```

返回

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);
```

返回

A set-returning function (SRF) that extracts the components of a geometry. It returns a set of **geometry_dump** rows, each containing a geometry (*geom* field) and an array of integers (*path* field).

For an atomic geometry type (POINT,LINESTRING,POLYGON) a single record is returned with an empty *path* array and the input geometry as *geom*. For a collection or multi-geometry a record is returned for each of the collection components, and the *path* denotes the position of the component inside the collection.

ST_Dump is useful for expanding geometries. It is the inverse of a **ST_Collect** / GROUP BY, in that it creates new rows. For example it can be use to expand MULTIPOLYGONS into POLYGONS.

版本: 2.0.0 支持多面体表面, 支持 TIN 表面。

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

This example breaks a PolyhedralSurface into its faces.

```
SELECT (a.p_geom).path[1] As path, ST_AsEWKT((a.p_geom).geom) As geom_ewkt
FROM (SELECT ST_Dump(ST_GeomFromEWKT('POLYHEDRALSURFACE(
((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1
1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)') ) AS p_geom ) AS a;
```

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

This example uses a TIN geometry.

```
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;
```

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](#) — [geometry_dump](#) rows with POINTs.

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 *n*th 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.

[geometry_dump](#): 2.0.0 [geometry_dump](#) rows, [ST_Dump](#) TIN [geometry_dump](#) rows.

1.5.0 [geometry_dump](#) rows.

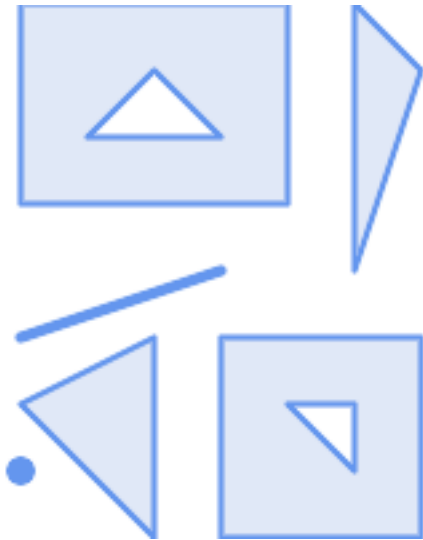
- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✔ This function supports 3d and will not drop the z-index.

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)

几何图形



```
SELECT path, ST_AsText(geom)
FROM (
  SELECT (ST_DumpPoints(g.geom)).*
  FROM
    (SELECT
      'GEOMETRYCOLLECTION(
        POINT ( 0 1 ),
        LINESTRING ( 0 3, 3 4 ),
```

```

        POLYGON (( 2 0, 2 3, 0 2, 2 0 )),
        POLYGON (( 3 0, 3 3, 6 3, 6 0, 3 0 ),
          ( 5 1, 4 2, 5 2, 5 1 )),
        MULTIPOLYGON (
          (( 0 5, 0 8, 4 8, 4 5, 0 5 )),
          ( 1 6, 3 6, 2 7, 1 6 )),
          (( 5 4, 5 8, 6 7, 5 4 ))
        )
      )::geometry AS geom
    ) AS g
  ) j;

```

path	st_astext
{1,1}	POINT(0 1)
{2,1}	POINT(0 3)
{2,2}	POINT(3 4)
{3,1,1}	POINT(2 0)
{3,1,2}	POINT(2 3)
{3,1,3}	POINT(0 2)
{3,1,4}	POINT(2 0)
{4,1,1}	POINT(3 0)
{4,1,2}	POINT(3 3)
{4,1,3}	POINT(6 3)
{4,1,4}	POINT(6 0)
{4,1,5}	POINT(3 0)
{4,2,1}	POINT(5 1)
{4,2,2}	POINT(4 2)
{4,2,3}	POINT(5 2)
{4,2,4}	POINT(5 1)
{5,1,1,1}	POINT(0 5)
{5,1,1,2}	POINT(0 8)
{5,1,1,3}	POINT(4 8)
{5,1,1,4}	POINT(4 5)
{5,1,1,5}	POINT(0 5)
{5,1,2,1}	POINT(1 6)
{5,1,2,2}	POINT(3 6)
{5,1,2,3}	POINT(2 7)
{5,1,2,4}	POINT(1 6)
{5,2,1,1}	POINT(5 4)
{5,2,1,2}	POINT(5 8)
{5,2,1,3}	POINT(6 7)
{5,2,1,4}	POINT(5 4)

(29 rows)

创建 TIN 面

This example uses a cube represented as a PolyhedralSurface.

```

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;

```


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)	

This example uses a Triangle geometry.

```
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;
```

path	wkt
-----+-----	
{1,1}	POINT(0 0)
{1,2}	POINT(0 9)
{1,3}	POINT(9 0)
{1,4}	POINT(0 0)

This example uses a TIN geometry.

```
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;
```

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` — `geometry_dump` table.

Synopsis

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



A set-returning function (SRF) that extracts the segments of a geometry. It returns a set of `geometry_dump` rows, each containing a geometry (*geom* field) and an array of integers (*path* field).

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

Availability: 3.2.0

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

几何集合

```
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)

几何集合, TIN 几何集合

This example uses a Triangle geometry.

```
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)

This example uses a TIN geometry.

```
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

This only works for POLYGON geometries, not MULTIPOLYGONS. Use `ST_Dump` to extract polygon elements from polygonal geometries: `ST_DumpRings((ST_Dump(geom)).geom )`

Availability: PostGIS 1.1.3. Requires PostgreSQL 7.3 or higher.



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



End point of a LineString

```
SELECT ST_AsText(ST_EndPoint('LINESTRING(1 1, 2 2, 3 3)::geometry));
```

POINT(3 3)

End point of a non-LineString is NULL

```
SELECT ST_EndPoint('POINT(1 1)::geometry) IS NULL AS is_null;
```

t

End point of a 3D LineString

3D endpoint.

```
SELECT ST_AsEWKT(ST_EndPoint('LINESTRING(1 1 2, 1 2 3, 0 0 5)'));
```

```
POINT(0 0 5)
```

```
ST_LineString [ ] ST_CircularString [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
```

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

POINT(6 3)

End point of a NURBSCurve

```
SELECT ST_AsText(ST_EndPoint('NURBSCURVE(2, (0 0, 1 1, 2 0))'::geometry));
```

POINT(2 0)



ST PointN, ST StartPoint

7.4.11 ST Envelope

ST Envelope —  (double precision; float8) .

Synopsis

```
geometry ST_Envelope(geometry g1);
```

返回

返回 float8 值。返回 ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)). (PostGIS 2 ZMIN/ZMAX 返回.)

返回 (xmin, ymin) 的 POLYGON 值, 返回 POINT 的 LINESTRING 值。

1.5.0 返回 float4 值。

✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1

✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.19

返回

```
SELECT ST_AsText(ST_Envelope('POINT(1 3)::geometry'));
```

```
POINT(1 3)
```

```
SELECT ST_AsText(ST_Envelope('LINESTRING(0 0, 1 3)::geometry'));
```

```
POLYGON((0 0,0 3,1 3,1 0,0 0))
```

```
SELECT ST_AsText(ST_Envelope('POLYGON((0 0, 0 1, 1.0000001 1, 1.0000001 0, 0 0))::geometry ←
));
```

```
POLYGON((0 0,0 1,1.0000001 1,1.0000001 0,0 0))
```

```
SELECT ST_AsText(ST_Envelope('POLYGON((0 0, 0 1, 1.0000000001 1, 1.0000000001 0, 0 0))':: ←
geometry));
```

```
POLYGON((0 0,0 1,1.0000000001 1,1.0000000001 0,0 0))
```

```
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;
```

POLYGON((20 75,20 150,125 150,125 75,20 75))

11

Box2D, Box3D, ST OrientedEnvelope

7.4.12 ST_ExteriorRing

ST ExteriorRing — .

Synopsis

```
geometry ST_ExteriorRing(geometry a_polygon);
```


POLYGON (exterior ring) . NULL



Note

XXXXXXXXXXXXXXXXXXXX. XXXXXXXX ST_Dump XXXXXXXXXXXXXXXX.

- ✔ 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.

☒☒

This example extracts exterior rings from 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))')
    ));
```

```
LINESTRING(0 0 1,1 1 1,1 2 1,1 1 1,0 0 1)
```

☒☒

[ST_InteriorRingN](#), [ST_Boundary](#), [ST_NumInteriorRings](#)

7.4.13 ST_GeometryN

ST_GeometryN — ST_Geometry ☒☒☒☒☒☒☒☒☒☒.

Synopsis

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

☒☒

Return the 1-based Nth element geometry of an input geometry which is a GEOMETRYCOLLECTION, MULTIPOINT, MULTILINESTRING, MULTICURVE, MULTIPOLYGON, or POLYHEDRALSURFACE. Otherwise, returns NULL.



Note

0.8.0 ☒☒☒☒☒☒☒ OGC ☒☒☒☒☒ 1-☒☒☒☒☒. ☒☒☒☒☒☒ 0-☒☒☒☒☒☒☒☒☒☒☒☒.

**Note**

ST_Dump 返回的几何体，是 ST_GeometryN 返回的几何体。

2.0.0 版本，支持 TIN 几何体。

2.0.0 版本，支持 NULL 几何体。2.0.0 版本，ST_GeometryN(..,1) 返回的几何体，是 ST_GeometryN 返回的几何体。

- ✓ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✓ This method implements the SQL/MM specification. SQL-MM 3: 9.1.5
- ✓ This function supports 3d and will not drop the z-index.
- ✓ This method supports Circular Strings and Curves.
- ✓ This function supports Polyhedral surfaces.
- ✓ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

提取

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);
```

提取，TIN 几何体

This example breaks a PolyhedralSurface 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;
```

Output:

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

This example uses a TIN geometry.

```
SELECT ST_AsEWKT(ST_GeometryN(geom,2)) as wkt
FROM
  (SELECT
    ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))
  )') AS geom
  ) AS g;
```

Output:

```
TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))
```

☒

ST_Dump, ST_NumGeometries

7.4.14 ST_GeometryType

ST_GeometryType — ST_Geometry 数据类型。

Synopsis

text **ST_GeometryType**(geometry g1);

☒

返回 'ST_LineString', 'ST_Polygon', 'ST_MultiPolygon' 等。返回的字符串与 ST_GeometryType(geometry) 返回的字符串相同。

版本: 2.0.0 引入 (polyhedral surface)。



This method implements the SQL/MM specification. SQL-MM 3: 5.1.4

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

例

```
SELECT ST_GeometryType(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
          ST_LineString
```

```
SELECT ST_GeometryType(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
) )'));
          ST_PolyhedralSurface
```

```
SELECT ST_GeometryType(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
) )'));
          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;
```

ST_Tin

例

GeometryType

7.4.15 ST_HasArc

ST_HasArc — Tests if a geometry contains a circular arc

Synopsis

boolean **ST_HasArc**(geometry geomA);

返回

如果 geomA 是圆弧，则返回 TRUE，否则返回 FALSE。

1.2.2 版本引入。

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

返回

```
SELECT ST_HasArc(ST_Collect('LINESTRING(1 2, 3 4, 5 6)', 'CIRCULARSTRING(1 1, 2 3, 4 5, 6 7, 5 6)'));
```

t

返回

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

7.4.16 ST_InteriorRingN

ST_InteriorRingN — 返回多边形内部环。

Synopsis

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

返回

返回第 N 个内部环。N 的范围是 1 到 (range) 内部环的数量。如果 N 为 NULL，则返回 NULL。



Note

如果 N 为 0，则返回外部环。使用 [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: 8.2.6, 8.3.5
- ✓ This function supports 3d and will not drop the z-index.

☒☒

```
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;
```

☒☒

[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);

返回

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.

返回

```
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));
```

返回

[ST_NumCurves](#), [ST_Dump](#), [ST_ExteriorRing](#), [ST_NumInteriorRings](#), [ST_NumGeometries](#)

7.4.19 ST_IsClosed

ST_IsClosed — LINESTRING 是否是闭合的。如果是，返回 TRUE。如果是开放的，返回 FALSE。如果是无效的几何体，返回 NULL。

Synopsis

boolean **ST_IsClosed**(geometry g);

返回

LINESTRING 是否是闭合的。如果是，返回 TRUE。如果是开放的，返回 FALSE。如果是无效的几何体，返回 NULL。



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.

版本: 2.0.0 函数名: `ST_IsClosed` (polyhedral surface) 函数名: `ST_IsClosed`.

✔ This function supports Polyhedral surfaces.

函数名: `ST_IsClosed`

```
SELECT ST_IsClosed('LINESTRING(0 0, 1 1)::geometry);
```

f

```
SELECT ST_IsClosed('LINESTRING(0 0, 0 1, 1 1, 0 0)::geometry);
```

t

```
SELECT ST_IsClosed('MULTILINESTRING((0 0, 0 1, 1 1, 0 0),(0 0, 1 1))::geometry);
```

f

```
SELECT ST_IsClosed('POINT(0 0)::geometry);
```

t

```
SELECT ST_IsClosed('MULTIPOINT((0 0), (1 1))::geometry);
```

t

函数名: `ST_IsClosed`

This example uses 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) ←
) )'));
```

t

This example uses the same cube with one side missing.

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

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

```
SELECT ST_IsCollection('LINestring(0 0, 1 1)::geometry);
```

f

```
SELECT ST_IsCollection('MULTIPOINT EMPTY)::geometry);
```

t

```
SELECT ST_IsCollection('MULTIPOINT((0 0))::geometry);
```

t

```
SELECT ST_IsCollection('MULTIPOINT((0 0), (42 42))::geometry);
```

t

```
SELECT ST_IsCollection('GEOMETRYCOLLECTION(POINT(0 0))::geometry);
```

t



ST_NumGeometries

7.4.21 ST_IsEmpty

ST_IsEmpty — Tests if a geometry is empty.

Synopsis

boolean **ST_IsEmpty**(geometry geomA);[illegible]

Note

SQL-MM ☒ ST_IsEmpty(NULL) ☒☒☒☒ 0 ☒☒☒☒☒☒☒, PostGIS ☒ NULL ☒☒☒☒☒☒.



✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
s2.1.1.1



✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.7



✔ This method supports Circular Strings and Curves.



Warning

```

PostGIS 2.0.0  ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)' )
PostGIS 2.0.0  , SQL/MM

```



```
SELECT ST_IsEmpty(ST_GeomFromText('GEOMETRYCOLLECTION EMPTY'));
```

t

```
SELECT ST_IsEmpty(ST_GeomFromText('POLYGON EMPTY'));
```

t

```
SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))'));
```

f

```
SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))')) = false;
```

```
t
(1 row)
```

```
SELECT ST_IsEmpty(ST_GeomFromText('CIRCULARSTRING EMPTY'));
```

```
t
```

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);

返回

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 新增。



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



This function supports M coordinates.

返回

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);



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 .



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**. 2.1.5.1



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.

例

```
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)
```

例

ST_IsClosed, ST_IsSimple, ST_StartPoint, ST_EndPoint

7.4.25 ST_IsSimple

ST_IsSimple — 如果几何体是简单的，则返回 TRUE，否则返回 FALSE。

Synopsis

boolean **ST_IsSimple**(geometry geomA);

例

如果几何体是简单的，则返回 TRUE，否则返回 FALSE。 该函数符合 OGC 简单要素实现规范，"OpenGIS 简单要素实现规范 (Ensuring OpenGIS compliance of geometries)" 的扩展。



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.

❏

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

f

```
SELECT ST_IsSimple(ST_GeomFromText('LINESTRING(1 1,2 2,2 3.5,1 3,1 2,2 1)'));
```

f

❏

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 M 坐标。M 坐标为 NULL 时返回 NULL。支持 3D 几何。



Note

此函数 (ST_M) 符合 OGC 简单要素实现规范 (OGC Simple Features Implementation Specification for SQL 1.1)，并符合 SQL/MM 规范 (SQL/MM specification)。此函数支持 3D 几何并保留 z-index。

- ✔ 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.

❏

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

4

❏

ST_GeomFromEWKT, ST_X, ST_Y, ST_Z

7.4.27 ST_MemSize

ST_MemSize — ST_Geometry 函数。

Synopsis

integer **ST_MemSize**(geometry geomA);

返回

ST_Geometry 函数。

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` 函数, TOAST 函数。

`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.

返回

```
--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 geomsizesize,
sum(ST_MemSize(geom))*1.00/pg_total_relation_size('public.neighborhoods')*100 As pergeom
FROM neighborhoods;
fulltable_size geomsizesize pergeom
-----
262144          96238          36.71188354492187500000
```

7.4.28 ST_NDims

ST_NDims — ST_Geometry 的维度。

Synopsis

integer **ST_NDims**(geometry g1);

返回

PostGIS 支持 2 - 2 维 (x,y), 3 - 3 维 (x,y,z), 3 - 4 维 (x,y,m), 4 - 4 维 (x,y,z,m) 的几何体。



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

返回

```
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
```

返回

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

7.4.29 ST_NPoints

ST_NPoints — 几何体的点数 (整数)。

Synopsis

integer **ST_NPoints**(geometry g1);

注意

此函数在 2.0.0 版本中引入。在 2.0.0 版本之前，此函数返回 0。

注意: 2.0.0 版本中引入 (polyhedral surface) 支持。



Note

1.3.4 版本中引入 (curve) 支持。1.3.4 版本之前，此函数返回 0。



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



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

示例

```
SELECT ST_NPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
29.07)'));
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)'));
4
```

示例

ST_NumPoints

7.4.30 ST_NRings

ST_NRings — 返回几何体中的环数。

Synopsis

integer **ST_NRings**(geometry geomA);

注意

此函数在 2.0.0 版本中引入。NumInteriorRings 函数返回内部环数，ST_NumPoints 函数返回总环数。



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 |          0
(1 row)
```



ST NumInteriorRings

7.4.31 ST_NumGeometries

ST NumGeometries — . .

Synopsis

```
integer ST_NumGeometries(geometry geom);
```



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

Version: 2.0.0, TIN

2.0.0 1 NULL 2.0.0 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).



This example returns 1; prior versions would have returned NULL.

```
SELECT ST_NumGeometries(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
```

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)))'));
```

3



ST_GeometryN, ST_Multi

7.4.32 ST_NumInteriorRings

ST_NumInteriorRings — 16-bit unsigned integer.

Synopsis

```
integer ST_NumInteriorRings(geometry a_polygon);
```



`XXXXXXXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXX NULL XXXXXX.`

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

Version: 2.0.0

This example counts the interior rings of a 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 — Integer . ST_NumInteriorRings Integer .

Synopsis

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


ST NumInteriorRings, ST PointN

7.4.34 ST_NumPatches

ST_NumPatches — 返回几何体中的面片数量。如果几何体为 NULL，则返回 NULL。

Synopsis

integer **ST_NumPatches**(geometry g1);

返回

返回几何体中的面片数量。如果几何体为 NULL，则返回 NULL。该函数与 ST_NumGeometries 类似，但 MM 返回的是面片数量。MM 返回的是 ST_NumGeometries 返回的几何体数量。

2.0.0 版本引入。

- ✓ This function supports 3d and will not drop the z-index.
- ✓ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✓ This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5
- ✓ This function supports Polyhedral surfaces.

返回

```
SELECT ST_NumPatches(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0) ←
    0)),
    ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0) ←
    ),
    ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
    ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1) ←
    ) )''));
6
```

返回

ST_GeomFromEWKT, **ST_NumGeometries**

7.4.35 ST_NumPoints

ST_NumPoints — ST_LineString 和 ST_CircularString 的几何体中的点数量。

Synopsis

integer **ST_NumPoints**(geometry g1);



```
ST_LineString geometry ST_CircularString geometry. 1.4 geometry
geometry. 1.4 geometry, geometry
ST_NPoints geometry. geometry ST_NPoints
geometry.
```

- ✔ This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**.
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 7.2.4


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


ST NPoints

7.4.36 ST_PatchN

ST PatchN — ST Geometry 

Synopsis

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



POLYHEDRALSURFACE, POLYHEDRALSURFACEM 1-N () . NULL . ST_GeometryN . ST_GeometryN .



Note!

Note

1-11111.



Note!

Note

XXXXXXXXXXXXXXXX ST Dump XXXXXXXXXXXXXXXX.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

- ✔ This method implements the SQL/MM specification. SQL/MM ISO/IEC 13249-3: 8.5
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.

**Note**

PostGIS 2.0.0 版本中，`ST_AsText` 函数对于 `LINESTRING` 类型的输入，如果输入是 `NULL`，则会返回 `NULL`。2.0.0 版本中，`ST_AsText` 函数对于 `LINESTRING` 类型的输入，如果输入是 `NULL`，则会返回 `NULL`。

PostGIS 2.3.0 版本中，`ST_AsText` 函数对于 `LINESTRING` 类型的输入，如果输入是 `NULL`，则会返回 `NULL`。

例

This example extracts all POINT geometries 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;
```

```
POINT(0 0)
POINT(1 1)
POINT(2 2)
(3 rows)
```

This example uses a CircularString.

```
SELECT ST_AsText(ST_PointN(ST_GeomFromText('CIRCULARSTRING(1 2, 3 2, 1 2)'), 2));
```

```
POINT(3 2)
```

```
SELECT ST_AsText(f)
FROM ST_GeomFromText('LINESTRING(0 0 0, 1 1 1, 2 2 2)') AS g
CROSS JOIN ST_PointN(g, -2) AS f; -- 1 based index
```

```
POINT Z (1 1 1)
```

例

ST_NPoints

7.4.38 ST_Points

`ST_Points` — 将 `LINESTRING` 类型的几何体转换为 `POINT` 类型的几何体。

Synopsis

geometry **ST_Points**(geometry geom);

返回

Returns a MultiPoint containing all the coordinates of a geometry. Duplicate points are preserved, including the start and end points of ring geometries. (If desired, duplicate points can be removed by calling [ST_RemoveRepeatedPoints](#) on the result).

To obtain information about the position of each coordinate in the parent geometry use [ST_DumpPoints](#). M and Z coordinates are preserved if present.



This method supports Circular Strings and Curves.



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

2.3.0 添加。

返回

```
SELECT ST_AsText(ST_Points('POLYGON Z ((30 10 4,10 30 5,40 40 6, 30 10))'));
MULTIPOINT Z ((30 10 4),(10 30 5),(40 40 6),(30 10 4))
```

返回

[ST_RemoveRepeatedPoints](#), [ST_PointN](#)

7.4.39 ST_StartPoint

ST_StartPoint — Returns the first point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

geometry **ST_StartPoint**(geometry geomA);

返回

Returns the first point of a LINESTRING, CIRCULARLINESTRING, or NURBSCURVE geometry as a POINT. For other geometries, returns the first point in coordinate order.



This method implements the SQL/MM specification. SQL-MM 3: 7.1.3



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



This method supports Circular Strings and Curves.

Note

Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString.



更新: 2.0.0 添加。PostGIS 2.0.0 添加。2.0.0 添加。NULL 添加。2.0 添加 NULL 添加。

例

Start M value of a LineString

```
SELECT ST_StartM('LINESTRING M(0 0 10, 1 1 20, 2 0 30)::geometry');
```

10

Start M value of a CircularString

```
SELECT ST_StartM('CIRCULARSTRING M(0 0 5, 1 1 10, 2 0 15)::geometry');
```

5

Start M value of a NURBSCurve

```
SELECT ST_StartM('NURBSCURVE M(2, (0 0 100, 1 1 200, 2 0 300))::geometry');
```

100

Geometry without M dimension returns NULL

```
SELECT ST_StartM('LINESTRING(0 0, 1 1)::geometry') IS NULL AS is_null;
```

t

例

ST_EndM, ST_SetStartM, ST_StartPoint

7.4.41 ST_EndM

ST_EndM — Returns the M coordinate of the last point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

```
float8 ST_EndM(geometry geom);
```

例

Returns the M coordinate of the last point of a LINESTRING, CIRCULARLINESTRING, or NURBSCURVE geometry. Returns NULL if the input geometry does not have an M dimension or is not a LINESTRING, CIRCULARLINESTRING, or NURBSCURVE.



This method supports Circular Strings and Curves.

☒☒

End M value of a LineString

```
SELECT ST_EndM('LINESTRING M(0 0 10, 1 1 20, 2 0 30)::geometry');
```

30

End M value of a CircularString

```
SELECT ST_EndM('CIRCULARSTRING M(0 0 5, 1 1 10, 2 0 15)::geometry');
```

15

End M value of a NURBSCurve

```
SELECT ST_EndM('NURBSCURVE M(2, (0 0 100, 1 1 200, 2 0 300))::geometry');
```

300

Geometry without M dimension returns NULL

```
SELECT ST_EndM('LINESTRING(0 0, 1 1)::geometry') IS NULL AS is_null;
```

t

☒☒

ST_StartM, ST_SetEndM, ST_EndPoint

7.4.42 ST_SetStartM

ST_SetStartM — Sets the M coordinate of the first point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

geometry **ST_SetStartM**(geometry geom, float8 m);

☒☒

Returns a copy of the input **LINESTRING**, **CIRCULARLINESTRING**, or **NURBSCURVE** geometry with the M coordinate of the first point set to the specified value. If the input geometry does not have an M dimension, it will be added. Returns NULL if the input is not a **LINESTRING**, **CIRCULARLINESTRING**, or **NURBSCURVE**.



This method supports Circular Strings and Curves.



Set start M value of a LineString

```
SELECT ST_AsText(ST_SetStartM('LINESTRING M(0 0 10, 1 1 20, 2 0 30)::geometry, 100));
```

```
LINESTRING M (0 0 100,1 1 20,2 0 30)
```

Add M dimension and set start M value

```
SELECT ST_AsText(ST_SetStartM('LINESTRING(0 0, 1 1, 2 0)::geometry, 50));
```

```
LINESTRING M (0 0 50,1 1 0,2 0 0)
```

Set start M value of a CircularString

```
SELECT ST_AsText(ST_SetStartM('CIRCULARSTRING M(0 0 5, 1 1 10, 2 0 15)::geometry, 99));
```

```
CIRCULARSTRING M (0 0 99,1 1 10,2 0 15)
```

Set start M value of a NURBSCurve

```
SELECT ST_AsText(ST_SetStartM('NURBSCURVE M(2, (0 0 100, 1 1 200, 2 0 300))::geometry, 500));
```

```
NURBSCURVE M (DEGREE 2,CONTROLPOINTS M (NURBSPOINT(WEIGHTEDPOINT M (0 0 500),WEIGHT 1),
NURBSPOINT(WEIGHTEDPOINT M (1 1 200),WEIGHT 1),NURBSPOINT(WEIGHTEDPOINT M (2 0 300),
WEIGHT 1)),KNOTS (KNOT(0,3),KNOT(1,3)))
```



ST_SetEndM, ST_StartM, ST_StartPoint

7.4.43 ST_SetEndM

ST_SetEndM — Sets the M coordinate of the last point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

geometry **ST_SetEndM**(geometry geom, float8 m);



Returns a copy of the input **LINESTRING**, **CIRCULARLINESTRING**, or **NURBSCURVE** geometry with the M coordinate of the last point set to the specified value. If the input geometry does not have an M dimension, it will be added. Returns **NULL** if the input is not a **LINESTRING**, **CIRCULARLINESTRING**, or **NURBSCURVE**.



This method supports Circular Strings and Curves.

- G: 点 (点) 点。
- S: 多边形 (多边形) 多边形。



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 多边形 (多边形) 多边形。

PostGIS: 2.0.0 多边形 (多边形) 多边形。

PostGIS: 2.1.0 点。多边形 (多边形) 多边形 S 多边形 (多边形) 多边形。

PostGIS: 2.2.0 多边形 TIN 多边形 (curve) 多边形 (多边形) 多边形。

点

```
SELECT ST_Summary(ST_GeomFromText('LINESTRING(0 0, 1 1)')) as geom,
       ST_Summary(ST_GeogFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) geog;
```

geom	geog
LineString[B] with 2 points	Polygon[BGS] with 1 rings
	ring 0 has 5 points
	:

(1 row)

```
SELECT ST_Summary(ST_GeogFromText('LINESTRING(0 0 1, 1 1 1)')) As geog_line,
       ST_Summary(ST_GeomFromText('SRID=4326;POLYGON((0 0 1, 1 1 2, 1 2 3, 1 1 1, 0 0 1))' ←
       ')) As geom_poly;
```

geog_line	geom_poly
LineString[ZBGS] with 2 points	Polygon[ZBS] with 1 rings
	: ring 0 has 5 points
	:

(1 row)

点

[PostGIS_DropBBox](#), [PostGIS_AddBBox](#), [ST_Force3DM](#), [ST_Force3DZ](#), [ST_Force2D](#), [geography](#)
[ST_IsValid](#), [ST_IsValid](#), [ST_IsValidReason](#), [ST_IsValidDetail](#)

7.4.45 ST_X

ST_X — Returns the X coordinate of a Point.

Synopsis

```
float ST_X(geometry a_point);
```

返回

返回 X 坐标。X 坐标为 NULL 时返回 NULL。返回类型与输入类型相同。



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.

返回

```
SELECT ST_X(ST_GeomFromEWKT('POINT(1 2 3 4)'));
```

```
1
```

```
SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
```

```
1.5
```

返回

ST_Centroid, **ST_GeomFromEWKT**, **ST_M**, **ST_XMax**, **ST_XMin**, **ST_Y**, **ST_Z**

7.4.46 ST_Y

ST_Y — Returns the Y coordinate of a Point.

Synopsis

float **ST_Y**(geometry a_point);

返回

返回 Y 坐标。Y 坐标为 NULL 时返回 NULL。返回类型与输入类型相同。



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)'));
```

2

```
SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
```

1.5

例

[ST_Centroid](#), [ST_GeomFromEWKT](#), [ST_M](#), [ST_X](#), [ST_YMax](#), [ST_YMin](#), [ST_Z](#)

7.4.47 ST_Z

ST_Z — Returns the Z coordinate of a Point.

Synopsis

```
float ST_Z(geometry a_point);
```

例

返回 Z 坐标。 Z 坐标为 NULL 时返回 NULL。 返回 Z 坐标。



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.

例

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

3

例

[ST_GeomFromEWKT](#), [ST_M](#), [ST_X](#), [ST_Y](#), [ST_ZMax](#), [ST_ZMin](#)

7.4.48 ST_ZMFlag

ST_ZMFlag — ST_Geometry 返回整数。

Synopsis

smallint **ST_ZMFlag**(geometry geomA);

返回

ST_Geometry 返回。

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.

返回

```
SELECT ST_ZMFlag(ST_GeomFromEWKT('LINESTRING(1 2, 3 4)'));
```

```
0
```

```
SELECT ST_ZMFlag(ST_GeomFromEWKT('LINESTRINGM(1 2 3, 3 4 3)'));
```

```
1
```

```
SELECT ST_ZMFlag(ST_GeomFromEWKT('CIRCULARSTRING(1 2 3, 3 4 3, 5 6 3)'));
```

```
2
```

```
SELECT ST_ZMFlag(ST_GeomFromEWKT('POINT(1 2 3 4)'));
```

```
3
```

返回

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

7.4.49 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)'));  
true
```

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



ST_ZMFlag

ST_HasM

7.4.50 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)'));
true
```

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

❏

ST_ZMFlag

ST_HasZ

7.4.51 ST_ControlPoints

ST_ControlPoints — Returns the control points of a NURBS curve as a MULTIPOINT geometry.

Synopsis

geometry **ST_ControlPoints**(geometry nurbscurve);

❏

Extracts the control points from a NURBS curve and returns them as a MULTIPOINT geometry. Control points define the control polygon that influences the shape of the NURBS curve.

The function raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0

❏

This example extracts control points from a NURBS curve.

```
SELECT ST_AsText(ST_ControlPoints('NURBS CURVE(2, (0 0, 5 10, 10 0))'::geometry));
```

```
MULTIPOINT((0 0),(5 10),(10 0))
```

This example counts the control points.

```
SELECT ST_NPoints(ST_ControlPoints('NURBS CURVE(2, (0 0, 5 10, 10 0))'::geometry));
```

```
3
```

❏

ST_MakeNurbsCurve, **ST_NumControlPoints**, **ST_Weights**

7.4.52 ST_Degree

ST_Degree — Returns the polynomial degree of a NURBS curve.

Synopsis

integer **ST_Degree**(geometry nurbscurve);



Returns the polynomial degree of a NURBS curve as an integer. The degree determines the curve's smoothness and flexibility. Higher degrees produce smoother curves but require more control points. The function raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0



This example returns the degree of a NURBS curve.

```
SELECT ST_Degree('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

```
2
```

This example checks whether a NURBS curve is cubic.

```
SELECT ST_Degree('NURBSCURVE(3, (0 0, 2 5, 5 8, 10 0))'::geometry) = 3;
```

```
t
```



[ST_MakeNurbsCurve](#), [ST_NumControlPoints](#)

7.4.53 ST_Weights

ST_Weights — Returns the weight array of a rational NURBS curve.

Synopsis

float8[] **ST_Weights**(geometry nurbscurve);



Returns the weight values of a rational NURBS curve as an array of double precision values. Weights control the influence of each control point on the curve shape. Higher weights pull the curve closer to the corresponding control point.

Returns NULL if the curve has no weights (non-rational B-spline) or if the input is NULL. Raises an error if the input is not a NURBS curve.

Availability: 3.7.0

☒☒

This example returns weights from a rational NURBS curve.

```
SELECT ST_Weights(ST_MakeNurbsCurve(2, 'LINESTRING(0 0, 5 10, 10 0)')::geometry, ARRAY[1.0, ←
    2.0, 1.0]);
```

```
{1,2,1}
```

This example checks the weights of a non-rational curve.

```
SELECT ST_Weights('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

```
null
```

☒☒

[ST_MakeNurbsCurve](#), [ST_NurbsCurveIsRational](#), [ST_ControlPoints](#)

7.4.54 ST_Knots

ST_Knots — Returns the knot vector of a NURBS curve.

Synopsis

```
float8[] ST_Knots(geometry nurbscurve);
```

☒☒

Returns the knot vector of a NURBS curve as an array of double precision values. The knot vector defines the parameter space and controls where and how the control points affect the curve. It must be a non-decreasing sequence with length equal to (number of control points + degree + 1).

Returns NULL if the curve uses the default uniform clamped knot vector or if the input is NULL. Raises an error if the input is not a NURBS curve.

Availability: 3.7.0

☒☒

This example returns knots from a NURBS curve with a custom knot vector.

```
SELECT ST_Knots(ST_MakeNurbsCurve(
    2,
    'LINESTRING(0 0, 5 10, 10 0)'::geometry,
    ARRAY[1.0, 2.0, 1.0],
    ARRAY[0,0,0,1,1,1]));
```

```
{0,0,0,1,1,1}
```

This example checks the knots of a curve using the default knot vector.

```
SELECT ST_Knots('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

```
null
```

￼￼

[ST_MakeNurbsCurve](#), [ST_Degree](#), [ST_NumControlPoints](#)

7.4.55 ST_NumControlPoints

ST_NumControlPoints — Returns the number of control points in a NURBS curve.

Synopsis

integer **ST_NumControlPoints**(geometry nurbscurve);

￼￼

Returns the number of control points in a NURBS curve. The number of control points must be at least (degree + 1) for a valid NURBS curve.

Raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0

￼￼

This example returns the number of control points.

```
SELECT ST_NumControlPoints('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

```
3
```

This example verifies that a curve has enough control points for its degree.

```
SELECT ST_NumControlPoints(curve)
>= ST_Degree(curve) + 1 AS valid
FROM (SELECT 'NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry AS curve) AS t;
```

```
t
```

￼￼

[ST_MakeNurbsCurve](#), [ST_ControlPoints](#), [ST_Degree](#)

7.4.56 ST_NurbsCurveIsRational

ST_NurbsCurveIsRational — Checks if a NURBS curve is rational (has weights).

Synopsis

boolean **ST_NurbsCurveIsRational**(geometry nurbscurve);

￼￼

Returns TRUE if the NURBS curve has weights assigned to its control points (rational NURBS), FALSE if it uses uniform weights (non-rational B-spline).

Rational NURBS curves provide more flexibility in curve shape control through variable weights. Non-rational B-splines use uniform weights and are a special case of NURBS curves.

Raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0

￼￼

This example checks whether a curve is rational.

```
SELECT ST_NurbsCurveIsRational(ST_MakeNurbsCurve(
    2,
    'LINESTRING(0 0, 5 10, 10 0)::geometry,
    ARRAY[1.0, 2.0, 1.0]));
```

t

This example checks a non-rational curve.

```
SELECT ST_NurbsCurveIsRational('NURBS CURVE(2, (0 0, 5 10, 10 0))::geometry');
```

f

￼￼

ST_MakeNurbsCurve, **ST_Weights**, **ST_IsNurbsCurve**

7.4.57 ST_IsNurbsCurve

ST_IsNurbsCurve — Checks if a geometry is a NURBS curve.

Synopsis

boolean **ST_IsNurbsCurve**(geometry geom);

￼￼

Returns TRUE if the geometry is a NURBS curve, FALSE otherwise. This function can be used to test geometry types before applying NURBS-specific functions.

Returns NULL if the input is NULL.

Availability: 3.7.0

☒☒

This example checks whether a geometry is a NURBS curve.

```
SELECT ST_IsNurbsCurve('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

t

This example tests a regular LineString.

```
SELECT ST_IsNurbsCurve('LINESTRING(0 0, 10 10)'::geometry);
```

f

☒☒

[ST_MakeNurbsCurve](#), [ST_GeometryType](#), [ST_NurbsCurveIsRational](#)

7.4.58 ST_Evaluate

ST_Evaluate — Evaluates a NURBS curve at a specific parameter value and returns the resulting point.

Synopsis

geometry **ST_Evaluate**(geometry nurbscurve, float8 parameter);

☒☒

Evaluates a NURBS curve at the specified parameter value and returns a POINT geometry representing the position on the curve. The parameter typically ranges from 0.0 (start of curve) to 1.0 (end of curve), though the actual range depends on the knot vector.

Raises an error if the input is not a NURBS curve or if evaluation fails. Returns NULL if either input is NULL.

Availability: 3.7.0

☒☒

This example returns the midpoint of a NURBS curve.

```
SELECT ST_AsText(ST_Evaluate('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry, 0.5));
```

This example samples multiple points along a curve.

```
SELECT t, ST_AsText(ST_Evaluate(curve, t)) AS point
FROM (SELECT 'NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry AS curve) AS c,
     generate_series(0, 1, 0.25) AS t;
```

☒☒

[ST_MakeNurbsCurve](#), [ST_NurbsToLineString](#)

7.4.59 ST_NurbsToLineString

ST_NurbsToLineString — Converts a NURBS curve to a LINESTRING by uniform sampling.

Synopsis

geometry **ST_NurbsToLineString**(geometry nurbscurve, integer num_segments=32);

☒☒

Converts a NURBS curve to a LINESTRING geometry by creating a piecewise linear approximation. The curve is sampled at uniformly distributed parameter values and the resulting points are connected with straight line segments.

The optional num_segments parameter specifies the number of line segments in the output (default: 32, range: 2-10000). Higher values produce smoother approximations but increase geometry complexity.

Raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0

☒☒

This example converts a NURBS curve to a LineString using the default quality.

```
SELECT ST_AsText(ST_NurbsToLineString('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry));
```

This example creates a high-quality approximation with 64 segments.

```
SELECT ST_AsText(ST_NurbsToLineString('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry, 64));
```

This example creates a lower-quality, faster approximation with 16 segments.

```
SELECT ST_AsText(ST_NurbsToLineString('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry, 16));
```

☒☒

[ST_MakeNurbsCurve](#), [ST_Evaluate](#), [ST_Segmentize](#)

7.5 简体中文 (editor)

7.5.1 ST_AddPoint

ST_AddPoint — 在几何体中添加点。

Synopsis

geometry **ST_AddPoint**(geometry linestring, geometry point);

geometry **ST_AddPoint**(geometry linestring, geometry point, integer position = -1);

☒☒

Adds a point to a LineString before the index *position* (using a 0-based index). If the *position* parameter is omitted or is -1 the point is appended to the end of the LineString.

1.1.0 简体中文.



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

☒☒

Add a point to the end of a 3D line

```
SELECT ST_AsEWKT(ST_AddPoint('LINESTRING(0 0 1, 1 1 1)', ST_MakePoint(1, 2, 3)));
```

```
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 简体中文.



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.

提取

Extract highest-dimension type:

```
SELECT ST_AsText(ST_CollectionExtract('GEOMETRYCOLLECTION( POINT(0 0), LINESTRING(1 1, 2 2) ↵
  '));
```

```
MULTILINESTRING((1 1, 2 2))
```

Extract points (type 1 == POINT):

```
SELECT ST_AsText(ST_CollectionExtract('GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(POINT(0 0)))',
  1));
```

```
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));
```

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

提取

[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);

返回

“返回” 返回。

- 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 返回。

返回

Single-element collection converted to an atomic geometry

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0))'));
```

```
POINT(0 0)
```

Nested single-element collection converted to an atomic geometry:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(MULTIPOINT((0 0)))'));
```

```
POINT(0 0)
```

Collection converted to a multi-geometry:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0),POINT(1 1))'));
```

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

```
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 50)), POLYGON ((90 50, 90 10, 50 10, 50 50, 90 50)))'));
```

```
MULTIPOLYGON(((10 50,50 50,50 10,10 10,10 50)),((90 50,90 10,50 10,50 50,90 50)))
```



[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 MULTI-SURFACE 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)')));

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)

```

This example uses 3D input.

```

SELECT ST_AsEWKT(ST_CurveToLine(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 ↵
150505 2,220227 150406 3)')));

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));

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
));

LINESTRING(0 0,49.99999999999996 -86.60254037844383,149.99999999999994 ↵
-86.6025403784439,200 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)'));
```

```
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).

❏

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

```
POINT(2 1)
```

❏

ST_SwapOrdinates

7.5.7 ST_Force2D

ST_Force2D — 将“2 维几何” 强制为 2D。

Synopsis

geometry **ST_Force2D**(geometry geomA);

❏

“2 维几何” 强制为 2D 几何。X 和 Y 坐标保持不变。OGC 2 维几何 (OGC 2 维几何) OGC 2 维几何。

版本: 2.0.0 强制为 (polyhedral surface) 2D。

版本: 2.1.0 开始, 在 2.0.x 版本中 ST_Force_2D 强制为 2D。

✓ This method supports Circular Strings and Curves.

✓ This function supports Polyhedral surfaces.

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

❏

```
SELECT ST_AsEWKT(ST_Force2D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
```

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

```
POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))
```

❏

ST_Force3D

7.5.8 ST_Force3D

ST_Force3D — 强制 XYZ 模式。ST_Force3DZ 类似。

Synopsis

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

描述

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.

版本: 2.0.0 支持 (polyhedral surface) 类型。

版本: 2.1.0 起, 2.0.x 版本使用 ST_Force_3D 函数。

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.

描述

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)')));
```

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

```
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))
```

描述

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);

函数

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

版本: 2.0.0 函数 (polyhedral surface) 函数.

版本: 2.1.0 函数, 在 2.0.x 函数 ST_Force_3DZ 函数.

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

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

函数

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)')));
```

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

```
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))
```

函数

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D

7.5.10 ST_Force3DM

ST_Force3DM — 函数 XYM 函数.

Synopsis

geometry **ST_Force3DM**(geometry geomA, float Mvalue = 0.0);

函数

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

版本: 2.1.0 函数, 在 2.0.x 函数 ST_Force_3DM 函数.

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)')));
```

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

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

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

```
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

ST_ForceCollection — .

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

XXXXXXXXXXXXXXXXXXXX. XXXX WKB XXXXXXXXXXXXXXXXXXXX.

: 2.0.0 (polyhedral surface) .

[illegible]

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

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

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

```
GEOMETRYCOLLECTION(POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1)))
```

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

```
GEOMETRYCOLLECTION(CIRCULARSTRING(220227 150406,220227 150407,220227 150406))
```

This example uses a PolyhedralSurface.

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

```

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

```

☒☒

[ST_AsEWKT](#), [ST_Force2D](#), [ST_Force3DM](#), [ST_Force3D](#), [ST_GeomFromEWKT](#)

7.5.13 ST_ForceCurve

`ST_ForceCurve` — 强制曲线, 强制曲线 (upcast) ☒☒.

Synopsis

geometry **ST_ForceCurve**(geometry g);

☒☒

强制曲线, 强制曲线. 强制曲线 (compoundcurve) ☒☒, 强制曲线 (multisurface) ☒☒☒☒. 强制曲线.

2.2.0 强制曲线.

✔ 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
)
);

```

```

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))

```

☒☒

[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);

返回

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 简体中文。

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

✔ This function supports M coordinates.

返回

[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);

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.

Synopsis

geometry **ST_Multi**(geometry geom);

返回

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

返回

```
SELECT ST_AsText(ST_Multi('POLYGON ((10 30, 30 30, 30 10, 10 10, 10 30))'));
```

```
MULTIPOLYGON(((10 30,30 30,30 10,10 10,10 30)))
```

返回

ST_AsText

7.5.21 ST_Normalize

ST_Normalize — 规范化几何对象。

Synopsis

geometry **ST_Normalize**(geometry geom);

返回

规范化几何对象。规范化几何对象，使其成为有效的几何对象。规范化几何对象，使其成为有效的几何对象。

规范化几何对象，使其成为有效的几何对象 (规范化几何对象)。

2.3.0 规范化几何对象。

返回

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

```
GEOMETRYCOLLECTION(POLYGON((0 0,0 10,10 10,10 0,0 0),(6 6,8 6,8 8,6 6),(2 2,4 2,4 4,2 2)),MULTILINESTRING((2 2,3 3),(0 0,1 1)),POINT(2 3))
```

￼￼

[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);
```

￼￼

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 ￼￼￼￼￼￼￼￼￼.

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)
```

￼￼

[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.

SQL

```
SELECT ST_AsText(ST_QuantizeCoordinates('POINT (100.123456 0)::geometry, 4));

POINT(100.12344360351562 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	010100000005f9a72083cdd5e405f9a72083cdd5e40123.456789123456)	POINT(123.456789123456 123.456789123456) ↵
14	010100000005f9a72083cdd5e405f9a72083cdd5e40123.456789123456)	POINT(123.456789123456 123.456789123456) ↵
13	010100000005e9a72083cdd5e405e9a72083cdd5e40123.45678912345599)	POINT(123.45678912345599 123.45678912345599) ↵
12	01010000000409a72083cdd5e40409a72083cdd5e40123.45678912345556)	POINT(123.45678912345556 123.45678912345556) ↵
11	0101000000009a72083cdd5e40009a72083cdd5e40123.45678912345466)	POINT(123.45678912345466 123.45678912345466) ↵
10	0101000000009872083cdd5e40009872083cdd5e40123.45678912344738)	POINT(123.45678912344738 123.45678912344738) ↵
9	0101000000008072083cdd5e40008072083cdd5e40123.45678912336007)	POINT(123.45678912336007 123.45678912336007) ↵
8	0101000000000070083cdd5e40000070083cdd5e40123.45678912103176)	POINT(123.45678912103176 123.45678912103176) ↵
7	0101000000000060083cdd5e40000060083cdd5e40123.4567891061306)	POINT(123.4567891061306 123.4567891061306) ↵
6	010100000000000083cdd5e40000000083cdd5e40123.45678901672363)	POINT(123.45678901672363 123.45678901672363) ↵
5	010100000000000003cdd5e40000000003cdd5e40123.456787109375)	POINT(123.456787109375 123.456787109375) ↵
4	010100000000000003cdd5e40000000003cdd5e40123.456787109375)	POINT(123.456787109375 123.456787109375) ↵
3	0101000000000000038dd5e400000000038dd5e40123.45654296875)	POINT(123.45654296875 123.45654296875) ↵
2	010100000000000000dd5e4000000000dd5e40123.453125 123.453125)	POINT(123.453125 123.453125) ↵
1	010100000000000000dc5e4000000000dc5e40123.4375 123.4375)	POINT(123.4375 123.4375) ↵
0	010100000000000000c05e4000000000c05e40123 123)	POINT(123 123) ↵
-1	01010000000000000005e4000000000005e40120 120)	POINT(120 120) ↵
-2	0101000000000000000584000000000005840POINT(96 96)	POINT(96 96) ↵
-3	0101000000000000000584000000000005840POINT(96 96)	POINT(96 96) ↵
-4	0101000000000000000584000000000005840POINT(96 96)	POINT(96 96) ↵
-5	0101000000000000000584000000000005840POINT(96 96)	POINT(96 96) ↵
-6	0101000000000000000584000000000005840POINT(96 96)	POINT(96 96) ↵
-7	0101000000000000000584000000000005840POINT(96 96)	POINT(96 96) ↵
-8	0101000000000000000584000000000005840POINT(96 96)	POINT(96 96) ↵
-9	0101000000000000000584000000000005840POINT(96 96)	POINT(96 96) ↵
-10	0101000000000000000584000000000005840POINT(96 96)	POINT(96 96) ↵
-11	0101000000000000000584000000000005840POINT(96 96)	POINT(96 96) ↵
-12	0101000000000000000584000000000005840POINT(96 96)	POINT(96 96) ↵
-13	0101000000000000000584000000000005840POINT(96 96)	POINT(96 96) ↵
-14	0101000000000000000584000000000005840POINT(96 96)	POINT(96 96) ↵
-15	0101000000000000000584000000000005840POINT(96 96)	POINT(96 96) ↵

(31 rows)

￼￼

[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);

ST_RemoveRepeatedPoints

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 简体中文.



This function supports Polyhedral surfaces.



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

Examples

```
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

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).

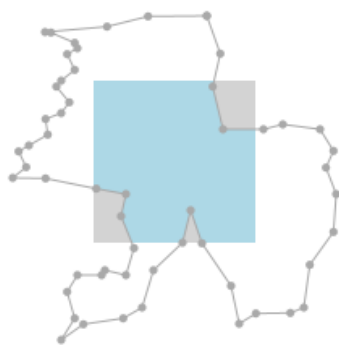
If `cartesian_hint` is set to `true`, the algorithm applies additional optimizations involving cartesian math to further reduce the resulting point number. Please note that using this option might introduce rendering artifacts if the resulting coordinates are projected into another (non-cartesian) coordinate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



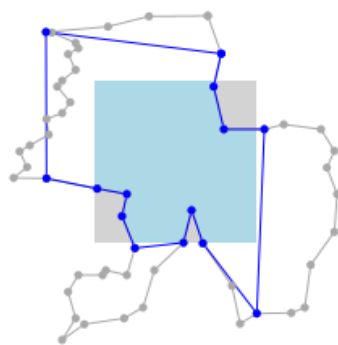
Warning

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`.

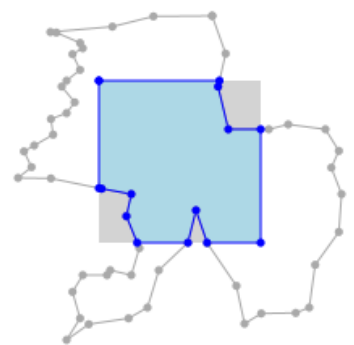
original
55 points



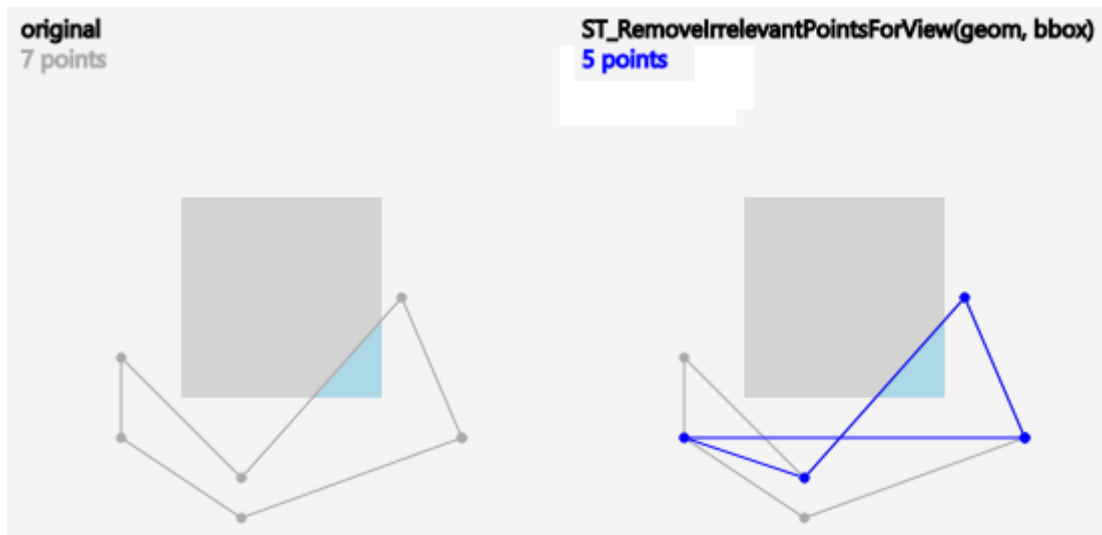
`ST_RemoveIrrelevantPointsForView(geom, bbox)`
15 points



`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.



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

☒☒

```
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));
```

```
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));
```

```
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));
```

```
LINESTRING
```

```
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));
```

```
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)));
```

```
POLYGON((0 30,30 30,30 0,0 0,0 30))
```

☒☒

[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));
```

```
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));
```

LINESTRING

7.5.28 ST_Reverse

ST_Reverse — 反转几何图形。

Synopsis

geometry **ST_Reverse**(geometry g1);

例

反转几何图形，返回反转后的几何图形。

Enhanced: 2.4.0 support for curves was introduced.

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

❏

```
SELECT ST_AsText(geom) as line, ST_AsText(ST_Reverse(geom)) As reverseline
FROM ST_MakeLine(ST_Point(1,2), ST_Point(1,10)) As geom;
```

line	reverseline
LINESTRING(1 2,1 10)	LINESTRING(1 10,1 2)

7.5.29 ST_Segmentize

ST_Segmentize — Returns a modified geometry/geography having no segment longer than a given distance.

Synopsis

geometry **ST_Segmentize**(geometry geom, float max_segment_length);
 geography **ST_Segmentize**(geography geog, float max_segment_length);

❏

Returns a modified geometry/geography having no segment longer than max_segment_length. Length is computed in 2D. Segments are always split into equal-length subsegments.

- For geometry, the maximum length is in the units of the spatial reference system.
- For geography, the maximum length is in meters. Distances are computed on the sphere. Added vertices are created along the spherical great-circle arcs defined by segment endpoints.



Note

This only shortens long segments. It does not lengthen segments shorter than the maximum length.



Warning

For inputs containing long segments, specifying a relatively short max_segment_length can cause a very large number of vertices to be added. This can happen unintentionally if the argument is specified accidentally as a number of segments, rather than a maximum length.

1.2.2 ST_Segmentize(geometry geom, float max_segment_length).

Enhanced: 3.0.0 Segmentize geometry now produces equal-length subsegments

Enhanced: 2.3.0 Segmentize geography now produces equal-length subsegments

❏: 2.1.0 ST_Segmentize(geometry geom, float max_segment_length).

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))

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

⚠️ 此函数在 2D 模式下运行，不支持 3D 几何体。如果输入几何体包含 Z 坐标，它们将被忽略。

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 snap 到网格上。

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);

1, 2, 3: (cell) (snap) .
 , NULL
 .

¶¶¶ 4: 1.1.0 ¶¶¶¶¶¶¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶ (¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶) ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶ 0 ¶¶¶¶¶¶¶¶¶¶.



Note

XXXXXXXXXXXXXXXXXXXXXXXX (ST_IsSimple XX).



Note

1.1.0 2 1.1.0, ,
 , . .

1.0.0RC1 ☐☐☐☐☐☐☐☐☐☐.

1.1.0 $\times \times \times \times$ Z $\times$ M $\times \times \times \times \times \times$.



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



This example snaps 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));
```

```
LINESTRING(1.112 2.123,4.111 3.237)
```

This example snaps a 4D geometry.

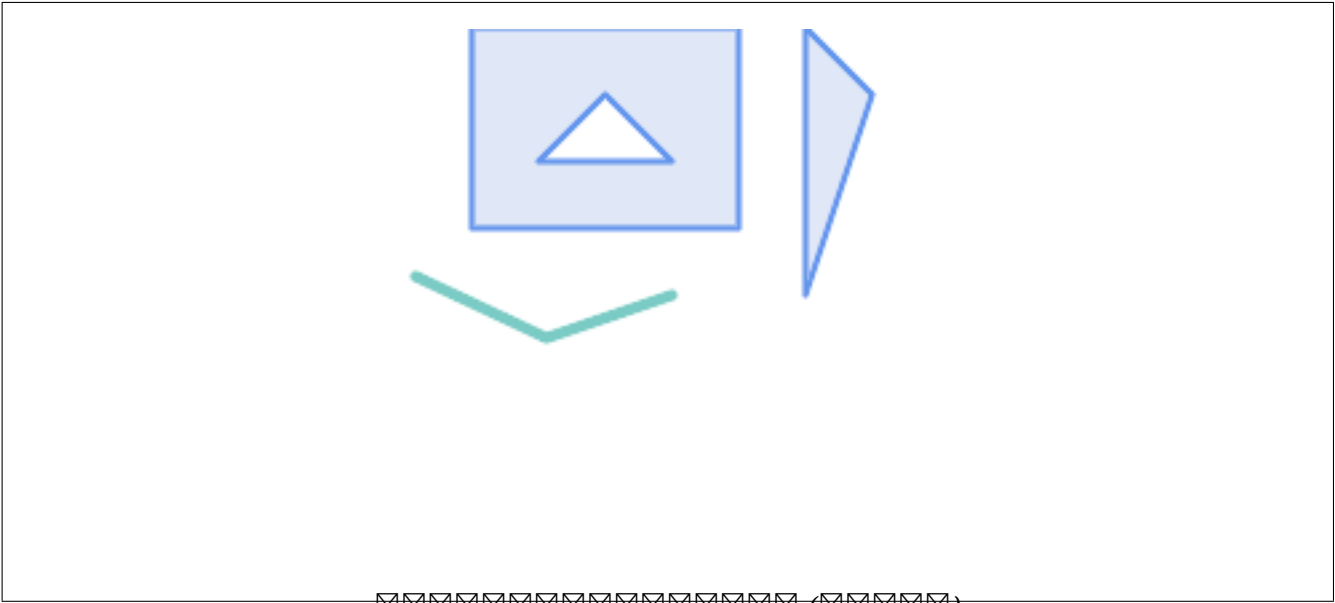
```
SELECT ST_AsEWKT(ST_SnapToGrid(ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 2.3456 1.11111,
4.111111 3.2374897 3.1234 1.1111, -1.11111112 2.123 2.3456 1.1111112)'),
ST_GeomFromEWKT('POINT(1.12 2.22 3.2 4.4444)'),
0.1, 0.1, 0.1, 0.01));
```

```
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));
```

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

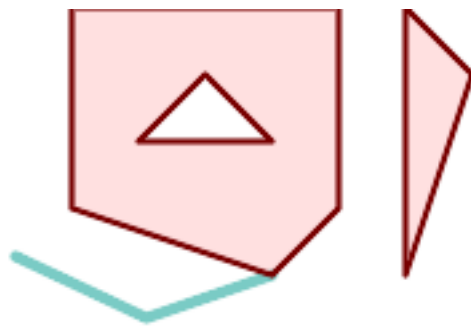


图 1.01 显示了从多边形到线段的距离。图中显示了从多边形到线段的距离。图中显示了从多边形到线段的距离。

```
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;

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)))
```

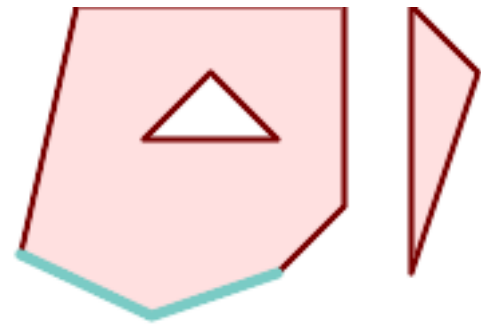


图 1.25 显示了从多边形到线段的距离。图中显示了从多边形到线段的距离。图中显示了从多边形到线段的距离。

```
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;

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)))
```


ords 返回一个包含 2 个元素的数组。 第一个元素 x, y, z, 第二个元素 m。

2.2.0 版本。

- ✔ 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

This example scales the 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;
```

```
POINT ZM (0 0 0 4)
```

例

[ST_FlipCoordinates](#)

7.6 Geometry Validation

7.6.1 ST_IsValid

ST_IsValid — Tests if a geometry is well-formed in 2D.

Synopsis

```
boolean ST_IsValid(geometry g);
boolean ST_IsValid(geometry g, integer flags);
```

例

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;
```

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);

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

This example returns the first three rejects from a successful quintuplet experiment.

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

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

```
--simple example
SELECT * FROM ST_IsValidDetail('LINESTRING(220227 150406,220227 150407,22020 150410)');
```

valid	reason	location
t		

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);
```

国国

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 国国国国国国

Availability: 1.4

Availability: 2.0 version taking flags.

国国

This example uses an invalid bow-tie polygon.

```
SELECT ST_IsValidReason(
  'POLYGON ((100 200, 100 100, 200 200,
    200 100, 100 200))':geometry) as validity_info;
```

```
Self-intersection[150 150]
```

This example returns the first three 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 default algorithm, and builds valid geometries by first extracting all lines, nodding that linework together, then building a value output from the linework. The requirement that no vertices are lost can generate complex collections as outputs.
- `"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 results tend to be more intuitive, where collapsed lines are discarded in the output.

The "keepcollapsed" key is only valid for the "structure" algorithm, and takes a value of "true" or "false".

- Keep collapsed of "false" is the default, and means that geometry components that collapse to a lower dimensionality, for example a one-point linestring will be dropped.
- Keep collapsed of "true" means geometry components that collapse to a lower dimensionality will be retained.

GEOS 

2.0.0 .

Enhanced: 2.0.1, speed improvements

Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT.

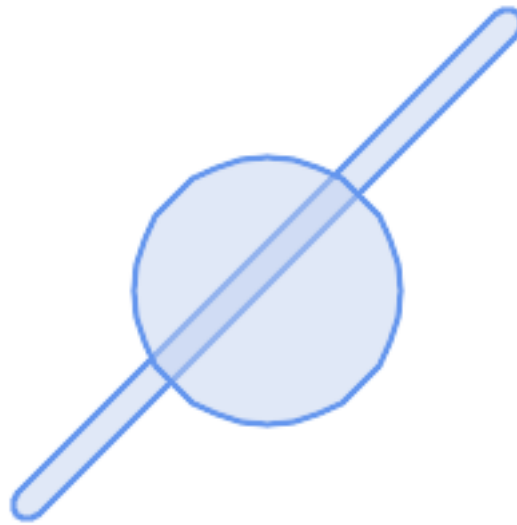
Enhanced: 3.1.0, added removal of Coordinates with NaN values.

Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS $\geq$ 3.10.0.

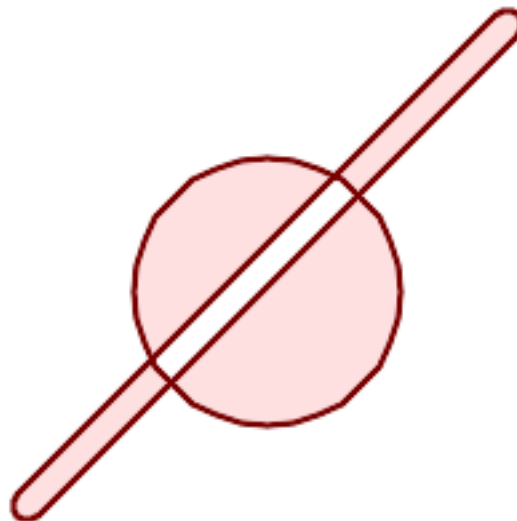


This function supports 3d and will not drop the z-index.

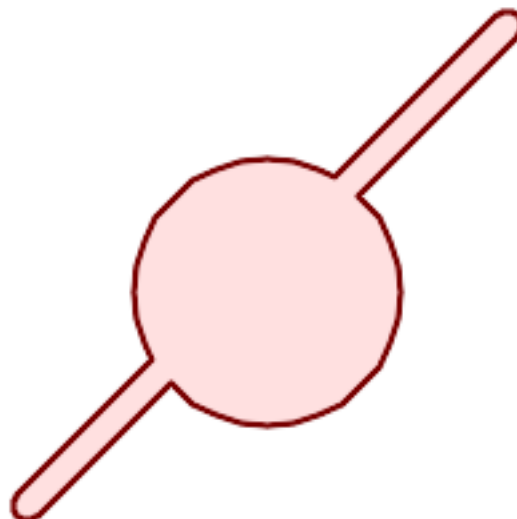




before_geom: MULTIPOLYGON of 2 overlapping polygons

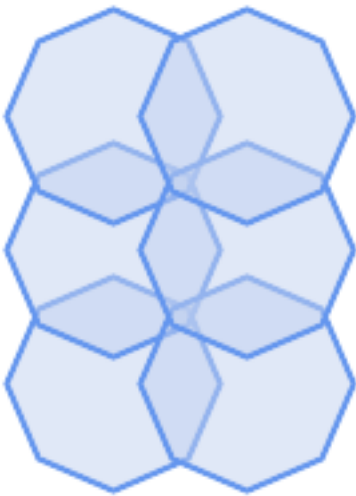


after_geom: MULTIPOLYGON of 4 non-overlapping polygons

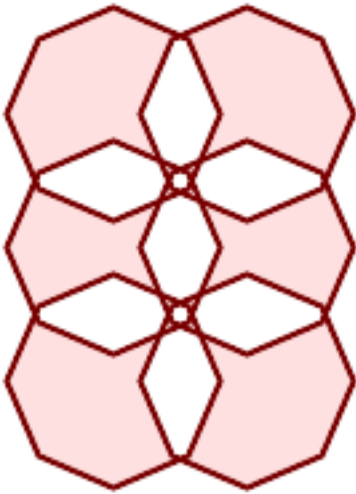


after_geom_structure: MULTIPOLYGON of 1 non-overlapping polygon

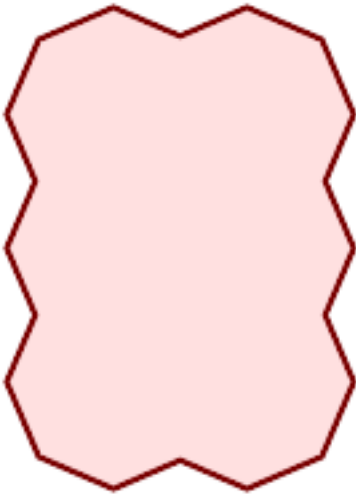
```
SELECT f.geom AS before_geom, ST_MakeValid(f.geom) AS after_geom, ST_MakeValid(f.geom, ↵  
    'method=structure') AS after_geom_structure  
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,  
191 195,192 195,193 194,194 194 193,195 192,195 191,
```



before_geom: MULTIPOLYGON of 6 overlapping polygons



after_geom: MULTIPOLYGON of 14 Non-overlapping polygons



after_geom_structure: MULTIPOLYGON of 1 Non-overlapping polygon

```
SELECT c.geom AS before_geom,  
       ST_MakeValid(c.geom) AS after_geom,  
       ST_MakeValid(c.geom, 'method=structure') AS after_geom_structure  
FROM (SELECT 'MULTIPOLYGON(((91 50,79 22,51 10,23 22,11 50,23 78,51 90,79 78,91 50)))
```

￼￼

```
SELECT ST_AsText(ST_MakeValid(
  'LINESTRING(0 0, 0 0)',
  'method=structure keepcollapsed=true'
));
```

```
POINT(0 0)
```

```
SELECT ST_AsText(ST_MakeValid(
  'LINESTRING(0 0, 0 0)',
  'method=structure keepcollapsed=false'
));
```

```
LINESTRING
```

￼￼

[ST_IsValid](#), [ST_Collect](#), [ST_CollectionExtract](#)

7.7 Spatial Reference System Functions

7.7.1 ST_InverseTransformPipeline

`ST_InverseTransformPipeline` — Return a new geometry with coordinates transformed to a different spatial reference system using the inverse of a defined coordinate transformation pipeline.

Synopsis

geometry **ST_InverseTransformPipeline**(geometry geom, text pipeline, integer to_srid);

￼￼

Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline to go in the inverse direction.

Refer to [ST_TransformPipeline](#) for details on writing a transformation pipeline.

Availability: 3.4.0

The SRID of the input geometry is ignored, and the SRID of the output geometry will be set to zero unless a value is provided via the optional `to_srid` parameter. When using [ST_TransformPipeline](#) the pipeline is executed in a forward direction. Using `ST_InverseTransformPipeline()` the pipeline is executed in the inverse direction.

Transforms using pipelines are a specialised version of [ST_Transform](#). In most cases `ST_Transform` will choose the correct operations to convert between coordinate systems, and should be preferred.

☒☒

Change WGS 84 long lat to UTM 31N using the EPSG:16031 conversion

This example transforms in the 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)
```

GDA2020 example.

This example uses 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)
```

☒☒

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 longitude/latitude.

```
SELECT ST_AsEWKT(ST_SetSRID(ST_Point(-123.365556, 48.428611),4326)) AS wgs84long_lat;
```

```
SRID=4326;POINT(-123.365556 48.428611)
```

Mark a point as WGS 84 longitude/latitude and then transform it to Web Mercator.

```
SELECT ST_AsEWKT(ST_Transform(ST_SetSRID(ST_Point(-123.365556, 48.428611),4326),3785)) AS sphere_merc; ↩
```

```
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));
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;
```

```
POLYGON((-71.1776848522251 42.3902896512903,-71.17768437663261 ↵
42.390382947801015,-71.17758443054647 42.390382667791854,-71.17758259272304 ↵
42.390289364798825,-71.1776848522251 42.3902896512903))
```

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));
```

```
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://boundless.github.io/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;
```

```
POLYGON((-140.9999840290591 73.42768865981984,-141 68,-170 68,-170.0000026915361 ↵
74.05492179637055,-140.9999840290591 73.42768865981984))
```

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 axiswap steps.

- Concatenated operations of the form: `urn:ogc:def:coordinateOperation,coordinateOperation:EPSG:XXXXXX`

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.

例

Change WGS 84 long lat to UTM 31N using the EPSG:16031 conversion

This example transforms in the forward direction.

```
SELECT ST_AsText(
  ST_TransformPipeline(
    'SRID=4326;POINT(2 49)::geometry',
    'urn:ogc:def:coordinateOperation:EPSG::16031'
  )
) AS utm_geom;
```

```
POINT(426857.9877165967 5427937.523342293)
```

This example transforms in the 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)
```

GDA2020 example.

This example uses `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)
```

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;
```

```
POINT(143.0000063280214 -36.999986718287545)
```

This example uses a PROJ pipeline definition matching EPSG:8447, as returned from projinfo. 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;

POINT(143.0000063280214 -36.999986718287545)
```

☐☐

[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') ORDER BY 1 LIMIT 10;
```

```
10156
10157
10158
10160
10162
10163
10164
10165
10166
10167
```

☐☐

[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`, `srtext`, `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
srtext    | PROJCS["NAD83 / BC Albers", ... ]
proj4text | +proj=aea +lat_0=45 +lon_0=-126 +lat_1=50 +lat_2=58.5 +x_0=1000000 +y_0=0 +
      datum=NAD83 +units=m +no_defs +type=crs
point_sw  | 0101000020E6100000E17A14AE476161C000000000000204840
point_ne  | 0101000020E610000085EB51B81E855CC0E17A14AE47014E40
```

☒☒

[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`, `srtext`, `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() ORDER BY auth_name, auth_srid
LIMIT 10;
```

auth_name	auth_srid	sname
EPSG	10156	ETRS89 + MSL UK & Ireland VORF08 depth
EPSG	10157	ETRS89 + CD UK & Ireland VORF08 depth
EPSG	10158	S34J-IRF
EPSG	10160	S34J reconstruction east-orientated
EPSG	10162	JGD2011 / Japan Plane Rectangular CS I + JGD2011 (vertical) height
EPSG	10163	JGD2011 / Japan Plane Rectangular CS II + JGD2011 (vertical) height
EPSG	10164	JGD2011 / Japan Plane Rectangular CS III + JGD2011 (vertical) height
EPSG	10165	JGD2011 / Japan Plane Rectangular CS IV + JGD2011 (vertical) height
EPSG	10166	JGD2011 / Japan Plane Rectangular CS V + JGD2011 (vertical) height
EPSG	10167	JGD2011 / Japan Plane Rectangular CS VI + JGD2011 (vertical) height

❏❏

[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+

7.8.1.3 ST_GeogFromText

ST_GeogFromText — WKT (几何) 转换为 geography 类型。

Synopsis

geography **ST_GeogFromText**(text EWKT);

几何

WKT 几何 WKT 几何。 指定 SRID 4326 几何。 ST_GeographyFromText 几何。 指定 SRID 4326 几何。

几何

This example converts longitude/latitude coordinates 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)'));
```

几何

ST_AsText, ST_GeographyFromText

7.8.1.4 ST_GeographyFromText

ST_GeographyFromText — WKT (几何) 转换为 geography 类型。

Synopsis

geography **ST_GeographyFromText**(text EWKT);

几何

WKT 几何 WKT 几何。 指定 SRID 4326 几何。

几何

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);
```

¶¶

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 - 2D SRID 0 (conformance suite) 2D.

WKT 2D (GEOMETRYCOLLECTION) 2D null 2D.



Note

2D WKT 2D, 2D. 2D 2D ST_GeomFromText 2D.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2](#)



This method implements the SQL/MM specification.

¶¶

```
SELECT ST_GeomCollFromText('GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(1 2, 3 4))');
```

¶¶

[ST_GeomFromText](#), [ST_SRID](#)

7.8.1.6 ST_GeomFromEWKT

ST_GeomFromEWKT — EWKT(Extended Well-Known Text) 2D ST_Geometry 2D.

Synopsis

```
geometry ST_GeomFromEWKT(text EWKT);
```

¶¶

OGC EWKT(Extended Well-Known Text) 2D PostGIS ST_Geometry 2D.



Note

EWKT 2D OGC 2D, SRID(2D) 2D PostGIS 2D.

2.0.0 (polyhedral surface) TIN.

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

```
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)');
```

This example uses a PolyhedralSurface.

```
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>
>'));
```

```
POINT(-4.5 54.25)
```

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>
>'));
```

```
POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1
52.38333333333333,13.1 52.65)) ↩
```

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"
>50000</subfield>
```



```

>E0130600</subfield>
                                <subfield code="e"
>E0133100</subfield>
                                <subfield code="f"
>N0523900</subfield>
                                <subfield code="g"
>N0522300</subfield>
                                </datafield>
                                <datafield tag="034" ind1=" " ind2=" ">
                                <subfield code="d"
>W004.500000</subfield>
                                <subfield code="e"
>W004.500000</subfield>
                                <subfield code="f"
>N054.250000</subfield>
                                <subfield code="g"
>N054.250000</subfield>
                                </datafield>
                                </record
>''));

```

```

GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667
52.38333333333333,13.1 52.38333333333333,13.1 52.65)),POINT(-4.5 54.25))

```

国国

ST_AsMARC21

7.8.1.8 ST_GeometryFromText

ST_GeometryFromText — WKT(Well-Known Text) 国国国国国国 ST_Geometry 国国国国国国. 国国国国
ST_GeomFromText 国国国国国国国国.

Synopsis

```

geometry ST_GeometryFromText(text WKT);
geometry ST_GeometryFromText(text WKT, integer srid);

```

国国

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.40

国国

ST_GeomFromText

7.8.1.9 ST_GeomFromText

ST_GeomFromText — WKT 国国国国国国国国国 ST_Geometry 国国国国国国.

Synopsis

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

ⓘ

OGC WKT(Well-Known Text) Ⓜ PostGIS ST_Geometry Ⓜ.



Note

ST_GeomFromText Ⓜ 2 Ⓜ, Ⓜ SRID Ⓜ Ⓜ Ⓜ (SRID=0) Ⓜ. Ⓜ SRID Ⓜ SRID Ⓜ.



This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**. Ⓜ 3.2.6.2 - Ⓜ SRID Ⓜ (conformance suite) Ⓜ.



This method implements the SQL/MM specification. SQL-MM 3: 5.1.40



This method supports Circular Strings and Curves.



Note

While 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

Ⓜ: PostGIS 2.0.0 Ⓜ ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') Ⓜ. PostGIS 2.0.0 Ⓜ, SQL/MM Ⓜ. Ⓜ ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') Ⓜ.

ⓘ

```
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,
```


✓ 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

例

```
SELECT ST_AsText(ST_LineFromText('LINESTRING(1 2, 3 4)')) AS aline,
       ST_LineFromText('POINT(1 2)') IS NULL AS null_return;
```

aline	null_return
LINESTRING(1 2,3 4)	t

例

ST_GeomFromText

7.8.1.11 ST_MLineFromText

ST_MLineFromText — WKT 转换为 ST_MultiLineString

Synopsis

geometry **ST_MLineFromText**(text WKT, integer srid);
 geometry **ST_MLineFromText**(text WKT);

例

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 (conformance suite).

WKT null 返回 null.



Note

WKT 必须是一个有效的 MultiLineString。如果 WKT 无效，ST_GeomFromText 将返回 null。

✓ 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))');
```

☐☐

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);
```

☐☐

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

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

WKT 3.2.6.2 null 3.2.6.2.



Note

☐☐ WKT 3.2.6.2 3.2.6.2, 3.2.6.2 3.2.6.2. 3.2.6.2 3.2.6.2 ST_GeomFromText 3.2.6.2.



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

☐☐

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

☐☐

ST_GeomFromText

7.8.1.13 ST_MPolyFromText

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

Synopsis

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

⌘⌘

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

OGC 3.2.6.2 - [⌘⌘⌘⌘ SRID ⌘⌘⌘⌘⌘⌘⌘](#) (conformance suite) [⌘⌘⌘⌘⌘⌘⌘⌘](#).

WKT [⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘](#).



Note

⌘⌘⌘ WKT [⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘](#), [⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘](#). [⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘](#) [⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘](#) ST_GeomFromText [⌘⌘⌘⌘⌘⌘](#).



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2](#)



This method implements the SQL/MM specification. SQL-MM 3: 9.6.4

⌘⌘

```
SELECT ST_MPolyFromText('MULTIPOLYGON(((0 0 1,20 0 1,20 20 1,0 20 1,0 0 1),(5 5 3,5 7 3,7 7 ←
3,7 5 3,5 5 3)))');
SELECT ST_MPolyFromText('MULTIPOLYGON((( -70.916 42.1002,-70.9468 42.0946,-70.9765 ←
42.0872,-70.9754 42.0875,-70.9749 42.0879,-70.9752 42.0881,-70.9754 42.0891,-70.9758 ←
42.0894,-70.9759 42.0897,-70.9759 42.0899,-70.9754 42.0902,-70.9756 42.0906,-70.9753 ←
42.0907,-70.9753 42.0917,-70.9757 42.0924,-70.9755 42.0928,-70.9755 42.0942,-70.9751 ←
42.0948,-70.9755 42.0953,-70.9751 42.0958,-70.9751 42.0962,-70.9759 42.0983,-70.9767 ←
42.0987,-70.9768 42.0991,-70.9771 42.0997,-70.9771 42.1003,-70.9768 42.1005,-70.977 ←
42.1011,-70.9766 42.1019,-70.9768 42.1026,-70.9769 42.1033,-70.9775 42.1042,-70.9773 ←
42.1043,-70.9776 42.1043,-70.9778 42.1048,-70.9773 42.1058,-70.9774 42.1061,-70.9779 ←
42.1065,-70.9782 42.1078,-70.9788 42.1085,-70.9798 42.1087,-70.9806 42.109,-70.9807 ←
42.1093,-70.9806 42.1099,-70.9809 42.1109,-70.9808 42.1112,-70.9798 42.1116,-70.9792 ←
42.1127,-70.979 42.1129,-70.9787 42.1134,-70.979 42.1139,-70.9791 42.1141,-70.9987 ←
42.1116,-71.0022 42.1273,
-70.9408 42.1513,-70.9315 42.1165,-70.916 42.1002)))',4326);
```

⌘⌘

[ST_GeomFromText](#), [ST_SRID](#)

7.8.1.14 ST_PointFromText

ST_PointFromText — [⌘⌘⌘ SRID ⌘⌘⌘ WKT ⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘⌘](#). SRID [⌘⌘⌘⌘⌘⌘⌘](#) [⌘⌘](#), [⌘⌘⌘⌘](#) 0 [⌘⌘⌘⌘](#).

Synopsis

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

☐☐

Constructs a PostGIS ST_Geometry point object from the OGC Well-Known text representation. If SRID is not given, it defaults to unknown (currently 0). If geometry is not a WKT point representation, returns null. If completely invalid WKT, then throws an error.



Note

ST_PointFromText 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_PointFromText 与 ST_MakePoint 和 ST_Point 类似。



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 格式字符串, 使用 `ST_GeomFromText` 函数。



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))');
```

```
010300000001000000050000006...
```

```
SELECT ST_PolygonFromText('POINT(1 2)') IS NULL AS point_is_notpoly;
```

```
t
```

SQL

ST_GeomFromText

7.8.1.16 ST_WKTToSQL

ST_WKTToSQL — WKT(Well-Known Text) 格式字符串 `ST_Geometry` 格式字符串。使用 `ST_GeomFromText` 函数。

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 格式字符串 `EWKB`(WKB) 格式字符串。

Synopsis

geography **ST_GeogFromWKB**(bytea wkb);

返回

ST_GeogFromWKB 将 WKB 格式的 PostGIS 的 WKB 数据转换为 geography 类型。该函数使用 SQL 的 Geometry Factory 函数。SRID 默认为 4326(WGS84 坐标系)。

 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\020
X\231C@\020X9\264\310~\\\\\300)\\\\\217\302\365\230C@' )
);
```

```
LINESTRING(-113.98 39.198, -113.981 39.195)
```

注意

ST_GeogFromText, **ST_AsBinary**

7.8.2.2 ST_GeomFromEWKB

ST_GeomFromEWKB — EWKB(Extended Well-Known Binary) 格式的 ST_Geometry 数据。

Synopsis

geometry **ST_GeomFromEWKB**(bytea EWKB);

返回

OGC EWKB(Extended Well-Known Binary) 格式的 PostGIS ST_Geometry 数据。



Note

EWKB 格式 OGC 的 SRID(坐标系) 默认为 PostGIS 的 SRID。

注意: 2.0.0 版本支持 (polyhedral surface) 和 TIN 数据。

-  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).

例

NAD83 坐标系 (SRID 4269) 的 LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932) 的 WKB 表示为:



Note

注意: 在 PostgreSQL 中, 标准字符串 (standard_conforming_strings) 默认为 on, 这导致在 AsEWKB 函数中, 单引号 (') 被转义为双单引号 (''), 而双引号 (") 则保持原样。这导致 AsEWKB 函数返回的 WKB 字符串与标准 SQL 的 WKB 字符串不一致。

```
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

在 PostgreSQL 中, standard_conforming_strings 默认为 on。您可以通过在单个查询或数据库或服务器级别更改默认值。以下是如何在 standard_conforming_strings = on 的情况下更改默认值的方法。在这种情况下, 我们使用标准 ANSI 单引号 (') 来转义单引号, 但斜杠 (/) 不需要转义。

```
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')');
```

例

ST_AsBinary, ST_AsEWKB, ST_GeomFromWKB

7.8.2.3 ST_GeomFromWKB

ST_GeomFromWKB — WKB(Well-Known Binary) 格式的二进制数据转换为 SRID 未知的几何对象。

Synopsis

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

例

ST_GeomFromWKB 函数接受 WKB 格式的二进制数据 (SRID 未知) 并返回一个几何对象。该函数是 SQL 函数 (Geometry Factory) 的一部分。ST_WKBToSQL 函数是 ST_GeomFromWKB 的逆函数。

SRID 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.

注意

Although bytea rep contains single \, these need to be escaped when inserting into a table. Unless standard_conforming_strings is set to on.

```
SELECT ST_AsEWKT(ST_GeomFromWKB(E' \001\002\000\000\000\002\000\000\000\037\205\353Q\270~\\\300\323Mb\020 \020\231C@\020X9\264\310~\\\300\\\217\302\365\230C@',4326)
);
```

```
SRID=4326;LINESTRING(-113.98 39.198,-113.981 39.195)
```

```
SELECT
  ST_AsText(ST_GeomFromWKB(ST_AsEWKB('POINT(2 5)::geometry)
    )
);
```

```
POINT(2 5)
```

注意

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

7.8.2.4 ST_LineFromWKB

[ST_LineFromWKB](#) — 将 SRID WKB 转换为 LINESTRING。

Synopsis

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

注意

[ST_LineFromWKB](#) 将 WKB SRID(标识 ID) 转换为 - 标识, LINESTRING - 标识。 标识 SQL (Geometry Factory) 标识。 SRID 标识, 标识 0 标识。 标识 bytea 标识, NULL 标识。



Note

OGC 3.2.6.2 - SRID (conformance suite)

**Note**

返回 `LINESTRING` 几何类型, 使用 `ST_GeomFromWKB` 函数。
 使用 `ST_GeomFromWKB` 函数, 返回 `LINESTRING` 几何类型。



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

例

```
SELECT ST_AsText(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
-----
LINESTRING(1 2,3 4) | t
```

例

`ST_GeomFromWKB`, `ST_LinestringFromWKB`

7.8.2.5 ST_LinestringFromWKB

`ST_LinestringFromWKB` — 使用 SRID 的 WKB 几何类型。

Synopsis

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

例

`ST_LinestringFromWKB` 使用 WKB 几何类型 SRID (几何类型 ID) 返回 `LINESTRING` 几何类型。
 使用 `ST_LinestringFromWKB` 函数, 返回 `LINESTRING` 几何类型。使用 `SQL` (Geometry Factory) 函数。

SRID 使用 `SRID`, 使用 0 表示。使用 `bytea` 返回 `LINESTRING` 几何类型, `NULL` 表示。
 使用 `SRID`。

**Note**

OGC 3.2.6.2 - 使用 SRID 的 WKB 几何类型 (conformance suite) 返回 `LINESTRING` 几何类型。

**Note**

返回 `LINESTRING` 几何类型, 使用 `ST_GeomFromWKB` 函数。
 使用 `ST_GeomFromWKB` 函数, 返回 `LINESTRING` 几何类型。

- ✓ 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

例

```
SELECT
  ST_AsText(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
LINESTRING(1 2,3 4)	t

例

[ST_GeomFromWKB](#), [ST_LineFromWKB](#)

7.8.2.6 ST_PointFromWKB

`ST_PointFromWKB` — 将 SRID 的 WKB 数据转换为点。

Synopsis

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

例

`ST_PointFromWKB` 将 WKB 数据转换为 SRID (默认 ID) 的 POINT 类型。SQL (Geometry Factory) 支持。SRID 默认为 0。bytea 类型，NULL 值。

- ✓ 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.

例

```
SELECT
  ST_AsText(ST_PointFromWKB(ST_AsEWKB('POINT(2 5)')::geometry)
);
```

POINT(2 5)

```
SELECT
  ST_AsText(ST_PointFromWKB(ST_AsEWKB('LINESTRING(2 5, 2 6)')::geometry)
);
```

null

例

ST_GeomFromWKB, ST_LineFromWKB

7.8.2.7 ST_WKBToSQL

ST_WKBToSQL — WKB(Well-Known Binary) 格式转换为 ST_Geometry 格式。 指定 SRID 使用 ST_GeomFromWKB 格式。

Synopsis

geometry **ST_WKBToSQL**(bytea WKB);

例

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

例

ST_GeomFromWKB

7.8.3 Other Formats

7.8.3.1 ST_Box2dFromGeoHash

ST_Box2dFromGeoHash — GeoHash 格式转换为 BOX2D 格式。

Synopsis

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

返回

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 与 GeoHash 字符串 BOX2D 字符串。 返回 BOX2D 字符串。

2.1.0 版本。

返回

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0');
```

```
BOX(-115.172816 36.114646,-115.172816 36.114646)
```

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 0);
```

```
BOX(-180 -90,180 90)
```

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10);
```

```
BOX(-115.17282128334 36.1146408319473,-115.172810554504 36.1146461963654)
```

返回

ST_GeoHash, ST_GeomFromGeoHash, ST_PointFromGeoHash

7.8.3.2 ST_GeomFromGeoHash

ST_GeomFromGeoHash — GeoHash 字符串。

Synopsis

geometry **ST_GeomFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

返回

GeoHash 字符串。 返回 GeoHash 字符串。

precision 字符串, ST_GeomFromGeoHash 与 GeoHash 字符串。

precision 字符串, ST_GeomFromGeoHash 与 GeoHash 字符串。

2.1.0 版本。

例

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0'));
```

```
POLYGON((-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646))
```

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 4));
```

```
POLYGON((-115.3125 36.03515625,-115.3125 36.2109375,-114.9609375 36.2109375,-114.9609375 36.03515625,-115.3125 36.03515625))
```

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10));
```

```
POLYGON((-115.17282128334 36.1146408319473,-115.17282128334 36.1146461963654,-115.172810554504 36.1146461963654,-115.172810554504 36.1146408319473,-115.17282128334 36.1146408319473))
```

例

ST_GeoHash, ST_Box2dFromGeoHash, ST_PointFromGeoHash

7.8.3.3 ST_GeomFromGML

ST_GeomFromGML — 将 GML 几何对象转换为 PostGIS 几何对象。

Synopsis

```
geometry ST_GeomFromGML(text geomgml);
geometry ST_GeomFromGML(text geomgml, integer srid);
```

例

OGC GML 几何对象 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。OGC GML 几何对象。

支持的 OGC GML 几何对象类型:

- GML 3.2.1 几何对象
- GML 3.1.1 几何对象 SF-2 (GML 3.1.0 和 3.0.0 几何对象)
- GML 2.1.2

OGC GML 规范: <http://www.opengeospatial.org/standards/gml>

1.5 几何对象。LibXML2 1.6 几何对象。

几何对象: 2.0.0 几何对象 (polyhedral surface) 和 TIN 几何对象。

几何对象: 2.0.0 几何对象 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 多面体 (MultiGeometry) 2D 3D 支持。PostGIS 支持 Z 坐标 ST_GeomFromGML 2D 支持。

GML 支持 SRS 支持。PostGIS 支持 ST_GeomFromGML 支持 SRS 支持。GML 支持 srsName 支持, 支持。

ST_GeomFromGML 支持 GML 支持。支持。支持 GML 支持 XLink 支持。



Note

ST_GeomFromGML 支持 SQL/MM 支持。

参数: srsName

```
SELECT ST_GeomFromGML($$
  <gml:LineString xmlns:gml="http://www.opengis.net/gml"
    srsName="EPSG:4269">
    <gml:coordinates>
      -71.16028,42.258729 -71.160837,42.259112 -71.161143,42.25932
    </gml:coordinates>
  </gml:LineString>
$$);
```

参数: XLink

```
SELECT ST_GeomFromGML($$
  <gml:LineString xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    srsName="urn:ogc:def:crs:EPSG::4269">
    <gml:pointProperty>
      <gml:Point gml:id="p1"
        ><gml:pos
          >42.258729 -71.16028</gml:pos
        </gml:Point>
      </gml:pointProperty>
      <gml:pos
        >42.259112 -71.160837</gml:pos>
      <gml:pointProperty>
        <gml:Point xlink:type="simple" xlink:href="#p1"/>
      </gml:pointProperty>
    </gml:LineString>
$$);
```

例: 创建多面体

```
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 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>
</gml:polygonPatches>
</gml:PolyhedralSurface
>'));
```

```
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);
```

☐☐

GeoJSON 格式数据转换为 PostGIS 几何类型。

ST_GeomFromGML JSON 格式 (geometry fragment) 格式数据。JSON 格式数据转换为 PostGIS 几何类型。

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 格式数据转换为 PostGIS 几何类型。JSON-C 0.9 格式数据转换为 PostGIS 几何类型。



Note

JSON-C 格式数据转换为 PostGIS 几何类型，格式数据转换为 PostGIS 几何类型。JSON-C 格式数据转换为 PostGIS 几何类型，"--with-jsondir=/path/to/json-c" 格式数据转换为 PostGIS 几何类型。格式数据转换为 PostGIS 几何类型 Section [2.2.3](#) 格式数据转换为 PostGIS 几何类型。



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

☐☐

```
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[-48.23456,20.12345]}')) AS wkt;
```

```
POINT(-48.23456 20.12345)
```

This example creates a 3D LineString.

```
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"LineString","coordinates":[[[1,2,3],[4,5,6],[7,8,9]]}')) AS wkt;
```

```
LINESTRING Z (1 2 3,4 5 6,7 8 9)
```


Synopsis

geometry **ST_GeomFromTWKB**(bytea twkb);

返回

`ST_GeomFromTWKB` 返回 TWKB("Tiny Well-Known Binary") 格式 (WKB) 的几何对象。

示例

```
SELECT ST_AsText(ST_GeomFromTWKB(ST_AsTWKB('LINESTRING(126 34, 127 35)::geometry')));
```

```
LINESTRING(126 34, 127 35)
```

```
SELECT ST_AsEWKT(ST_GeomFromTWKB(E'\\x620002f7f40dbce4040105'));
```

```
LINESTRING(-113.98 39.198,-113.981 39.195)
```

返回

ST_AsTWKB

7.8.3.7 ST_GMLToSQL

`ST_GMLToSQL` — GML 几何对象 `ST_Geometry` 的 SQL 表示。返回 `ST_GeomFromGML` 返回的几何对象。

Synopsis

geometry **ST_GMLToSQL**(text geomgml);
geometry **ST_GMLToSQL**(text geomgml, integer srid);

返回

 This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (多面体表面)
1.5 多面体表面。 LibXML2 1.6 多面体表面。
返回: 2.0.0 多面体表面 (polyhedral surface) 的 TIN 表示。
返回: 2.0.0 多面体表面 SRID 的表示。

返回

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

7.8.3.8 ST_LineFromEncodedPolyline

`ST_LineFromEncodedPolyline` — 编码多线 (polyline) 的几何对象。

Synopsis

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

返回

返回的几何体是 LineString 类型。

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.

参考: <http://developers.google.com/maps/documentation/utilities/polylinealgorithm>

2.2.0 版本引入。

示例

This example creates a LineString from a polyline.

```
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@'));
```

```
SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)
```

This example selects a different precision than the one used for polyline encoding.

```
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@',6));
```

```
SRID=4326;LINESTRING(-12.02 3.85,-12.095 4.07,-12.6453 4.3252)
```

参考

ST_AsEncodedPolyline

7.8.3.9 ST_PointFromGeoHash

ST_PointFromGeoHash — GeoHash 字符串转换为点。

Synopsis

point **ST_PointFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

返回

GeoHash 字符串。返回的 GeoHash 字符串与输入字符串相同。

precision 指定要保留的十进制位数，ST_PointFromGeoHash 返回的 GeoHash 字符串的精度与 precision 相同。

precision 指定要保留的十进制位数，ST_PointFromGeoHash 返回的 GeoHash 字符串的精度与 precision 相同。

2.1.0 版本引入。

☐☐

```
SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0'));
```

```
POINT(-115.172816 36.114646)
```

```
SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 4));
```

```
POINT(-115.13671875 36.123046875)
```

```
SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10));
```

```
POINT(-115.172815918922 36.1146435141563)
```

☐☐

ST_GeoHash, ST_Box2dFromGeoHash, ST_GeomFromGeoHash

7.8.3.10 ST_FromFlatGeobufToTable

ST_FromFlatGeobufToTable — Creates a table based on the structure of FlatGeobuf data.

Synopsis

void **ST_FromFlatGeobufToTable**(text schemaname, text tablename, bytea FlatGeobuf input data);

☐☐

Creates a table based on the structure of FlatGeobuf data. (<http://flatgeobuf.org>).

schema Schema name.

table Table name.

data Input FlatGeobuf data.

Availability: 3.2.0

7.8.3.11 ST_FromFlatGeobuf

ST_FromFlatGeobuf — Reads FlatGeobuf data.

Synopsis

setof anyelement **ST_FromFlatGeobuf**(anyelement Table reference, bytea FlatGeobuf input data);

☐☐

Reads FlatGeobuf data (<http://flatgeobuf.org>). NOTE: PostgreSQL bytea cannot exceed 1GB.

tabletype reference to a table type.

data input FlatGeobuf data.

Availability: 3.2.0

7.9 Geometry Output

7.9.1 Well-Known Text (WKT)

7.9.1.1 ST_AsEWKT

ST_AsEWKT — Returns WKT(Well-Known Text) representation of the geometry prefixed with the SRID.

Synopsis

```
text ST_AsEWKT(geometry g1);
text ST_AsEWKT(geometry g1, integer maxdecimaldigits=15);
text ST_AsEWKT(geography g1);
text ST_AsEWKT(geography g1, integer maxdecimaldigits=15);
```

Warning

Returns the Well-Known Text representation of the geometry prefixed with the SRID. The optional *maxdecimaldigits* argument may be used to reduce the maximum number of decimal digits after floating point used in output (defaults to 15).

To perform the inverse conversion of EWKT representation to PostGIS geometry use [ST_GeomFromEWKT](#).



Warning

Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use [ST_ReducePrecision](#) with a suitable gridsize first.



Note

The WKT spec does not include the SRID. To get the OGC WKT format use [ST_AsText](#).



Warning

WKT format does not maintain precision so to prevent floating truncation, use [ST_AsBinary](#) or [ST_AsEWKB](#) format for transport.

Enhanced: 3.1.0 support for optional precision parameter.

2.0.0: [ST_AsEWKT](#), [ST_AsText](#), [ST_AsTIN](#).



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).

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.

[illegible]

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

```
SELECT ST AsText('POINT(111.111111 1.111111)')::geometry);
```

POINT(111.1111111 1.1111111)

```
SELECT ST_AsText('POINT(111.111111 1.111111)'::geometry, 2);
```

POINT(111.11 1.11)



ST AsBinary, ST AsEWKB, ST AsEWKT, ST GeomFromText

7.9.2 Well-Known Binary (WKB)

7.9.2.1 ST_AsBinary

ST_AsBinary — Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.

Synopsis

```
bytea ST_AsBinary(geometry g1);
bytea ST_AsBinary(geometry g1, text NDR_or_XDR);
bytea ST_AsBinary(geography g1);
bytea ST_AsBinary(geography g1, text NDR or XDR);
```



Returns the OGC/ISO **Well-Known Binary** (WKB) representation of the geometry. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding: either 'NDR' for little-endian; or 'XDR' for big-endian. Supplying unknown arguments will result in little-endian output.

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

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



Note

The OGC/ISO WKB format does not include the SRID. To get the EWKB format which does include the SRID use **ST_AsEWKB**



Note

PostgreSQL outputs bytea values in hex encoding by default. If your GUI tools require the old behavior, then SET `bytea_output='escape'` in your database.

Version: 2.0.0, TIN

□□□□: 2.0.0 □□□□□□□□□□□□□□□□□□□□.

[illegible]

1.5.0 ☒☒☒☒☒☒☒☒☒☒☒☒.

Changed: 2.0.0 Inputs to this function can not be unknown -- must be geometry. Constructs such as `ST_AsBinary('POINT(1 2)')` are no longer valid and you will get an `n ST_AsBinary(unknown) is not unique` error. Code like that needs to be changed to `ST_AsBinary('POINT(1 2)::geometry');`. If that is not possible, then install `legacy.sql`.



✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
s2.1.1.1



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



✔ This method supports Circular Strings and Curves.



✔ This function supports Polyhedral surfaces.



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



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



```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

[illegible]

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326), 'XDR');
```

[illegible]

ST_GeomFromWKB, ST_AsEWKB, ST_AsTWKB, ST_AsText,

7.9.2.2 ST_AsEWKB

ST_AsEWKB — Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.

Synopsis

```
bytea ST_AsEWKB(geometry g1);
bytea ST_AsEWKB(geometry g1, text NDR_or_XDR);
```



Returns the **Extended Well-Known Binary** (EWKB) representation of the geometry with SRID metadata. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding: either 'NDR' for little-endian; or 'XDR' for big-endian. Supplying unknown arguments will result in little-endian output.

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

To perform the inverse conversion of EWKB to PostGIS geometry use `ST_GeomFromEWKB`.



Note

To get the OGC/ISO WKB format use `ST_AsBinary`. Note that OGC/ISO WKB format does not include the SRID.

Version: 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).


```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
\x0103000020e6100000010000000500000000000000000000000000000000000000000000000000000000000000000000000f03f000000000000f03f000000000000f03f000000000000f03f000000000000000000000000000000000000000000000000
```

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326), 'XDR');
```

[illegible]

ST_AsBinary, ST_GeomFromEWKB, ST_SRID

7.9.2.3 ST_AsHEXEWKB

ST AsHEXEWKB — $\square\square\square\square\square\square$ (NDR) $\square\square\square\square\square\square$ (XDR) $\square\square\square\square\square\square$ HEXEWKB ($\square\square\square$) $\square\square\square\square$
 $\square\square\square\square\square$.

Synopsis

```
text ST_AsHEXEWKB(geometry g1, text NDRorXDR);
text ST_AsHEXEWKB(geometry g1);
```


XXXXXXXXXX (NDR) XXXXX (XDR) XXXXXX HEXEWKB (XXX) XXXXXXXXXXXX. XXXXX
XXXXXXXXXX NDR XXXXX.



Note

1.2.2 $\square\square\square\square\square\square\square\square\square\square\square$.



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



This method supports Circular Strings and Curves.


```
SELECT ST_AsHEXEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

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

Which gives the same answer as:

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

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。

参数

参数

```
SELECT ST_AsEncodedPolyline(GeomFromEWKT('SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)'));
```

```
_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。

This example builds the SQL for a Boston-to-San-Francisco path with segments every 100 kilometers.

```
SELECT ST_AsEncodedPolyline(ST_Segmentize(
    ST_GeogFromText('LINESTRING(-71.0519 42.4935,-122.4483 37.64)'),
    100000)::geometry) As encodedFlightPath;
```

在 HTML 页面中嵌入 Polyline 数据。

```
<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>
```

国国

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);
```

国国

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);
```

国国

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 国国国国国国国国国国国国.

☒☒

```
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;
```

```
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.

When the input record includes duplicate column names, the resulting "properties" object will contain repeated keys. `ST_AsGeoJSON` raises a warning in that case because PostgreSQL jsonb keeps only the last value for duplicate keys.

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 支持空值。

1.5.0 支持空值。

更新: 2.0.0 支持空值 (default arg) 支持空值 (named arg) 支持空值。

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

Changed: 3.7.0 added warning about duplicate keys



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

更新

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);
```

```
{"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;
```

```
{"type": "MultiLineString", "coordinates": [[[-89.734634999999997, 31.492072000000000], [-89.734955999999997, 31.492237999999997]]]}
```

3D geometries are supported:

```
SELECT ST_AsGeoJSON('LINESTRING(1 2 3, 4 5 6)');
```

```
{"type": "LineString", "coordinates": [[1, 2, 3], [4, 5, 6]]}
```

Options argument can be used to add BBOX and CRS in GeoJSON output:

```
SELECT ST_AsGeoJSON(ST_SetSRID('POINT(1 1)::geometry', 4326), 9, 4|1);
```

```
{"type": "Point", "crs": {"type": "name", "properties": {"name": "urn:ogc:def:crs:EPSG::4326"}}, "bbox": [1.000000000, 1.000000000, 1.000000000, 1.000000000], "coordinates": [1, 1]}
```

☒☒

ST_GeomFromGeoJSON, **ST_ForcePolygonCCW**, **ST_Transform**

7.9.3.5 ST_AsGML

ST_AsGML — ☒☒☒ GML 2 ☒☒ GML 3 ☒☒☒☒☒☒☒☒☒.

Synopsis

```
text ST_AsGML(geometry geom, integer maxdecimaldigits=15, integer options=0);
text ST_AsGML(geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);
text ST_AsGML(integer version, geometry geom, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);
text ST_AsGML(integer version, geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);
```

☒☒

Return the geometry as a Geography Markup Language (GML) element. The version parameter, if specified, may be either 2 or 3. If no version parameter is specified then the default is assumed to be 2. The maxdecimaldigits argument may be used to reduce the maximum number of decimal places used in output (defaults to 15).

**Warning**

Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use **ST_ReducePrecision** with a suitable gridsize first.

GML 2 的 2.1.2 版本, GML 3 的 3.1.1 版本。

' 的 (bitfield) 。 CRS 的 GML 的, 的/的 的。

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

的 (的) 的, 的 的 NULL 的'gml' 的。

1.3.2 的。

1.5.0 的。

的: 2.0.0 的。 的 GML 3 的 的'4' 的。 GML 3 的 TIN 的。 的'32' 的 的。

的: 2.0.0 的 (named arg) 的。

的: 2.1.0 的 GML 3 的 ID 的。

**Note**

ST_AsGML 的 3 的 TIN 的。

- ✓ 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).

的: 的 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 srsDimension="2"
>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" srsDimension="2"
><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" srsDimension="2"
><gml:lowerCorner
>2 1</gml:lowerCorner
><gml:upperCorner
>20 10</gml:upperCorner
></gml:Envelope>
```

This example uses a PolyhedralSurface.

```
SELECT ST_AsGML(
3,
ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ));
```

```

st_asgml
-----
<gml:PolyhedralSurface>
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList>
        </gml:LinearRing>
      </gml:exterior>
    </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing>
          <gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing>
          <gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing>
          <gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing>
          <gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 0 0 1 0</gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing>
          <gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
    </gml:polygonPatches>
  </gml:PolyhedralSurface>

```

☒☒

ST_GeomFromGML

7.9.3.6 ST_AsKML

ST_AsKML — 将 GML 2 或 GML 3 几何体转换为 KML。

Synopsis

```
text ST_AsKML(geometry geom, integer maxdecimaldigits=15, text nprefix=NULL);
text ST_AsKML(geography geog, integer maxdecimaldigits=15, text nprefix=NULL);
```

注意

将 KML (Keyhole Markup Language) 几何体转换为 KML。将 KML 几何体转换为 KML。将 KML 几何体转换为 KML (精度 15)，将 KML 几何体转换为 KML 2 或 KML 3 几何体。



Warning

Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use **ST_ReducePrecision** with a suitable gridsize first.



Note

PostGIS 的 Proj 库版本为 4.9.0。Proj 库版本为 4.9.0 的 PostGIS_Full_Version 版本。



Note

1.2.2 版本。1.3.2 版本。



Note

2.0.0 版本。2.0.0 版本。



Note

Changed: 3.0.0 - Removed the "versioned" variant signature



Note

ST_AsKML 的 SRID 值。



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 — 将几何体转换为经纬度文本格式。

Synopsis

text **ST_AsLatLonText**(geometry pt, text format=“”);

例

将点几何体转换为经纬度文本格式。



Note

经纬度文本格式/坐标格式。格式为 X(经度) Y(纬度) 格式。经度 (经度 -180 经度 180 度, 纬度 -90 纬度 90 度) 格式。

经纬度文本格式/坐标格式。格式为 X(经度) Y(纬度) 格式。经度 (经度 -180 经度 180 度, 纬度 -90 纬度 90 度) 格式。格式为 “D”, 格式为 “M”, 格式为 “S”, 格式为 (格式, cardinal direction) 格式 “C” 格式。D, M, S 格式 (“SSS.SSSS” 格式 “1.0023” 格式)。

M, S, C 格式。 “C” 格式, 格式为 “-” 格式。 “S” 格式, 格式为 (格式, cardinal direction) 格式。 “M” 格式, 格式为 (格式, cardinal direction) 格式。

经纬度文本格式 (格式 0 度) 格式。

2.0 格式。

例

例

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)'));
```

```
2°19'29.928"S 3°14'3.243"W
```

(例) 例。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D°M'S.SSS"C'));
```

```
2°19'29.928"S 3°14'3.243"W
```

D, M, S, C 例。 例。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D degrees, M minutes, S seconds to  
the C'));
```

```
2 degrees, 19 minutes, 30 seconds to the S 3 degrees, 14 minutes, 3 seconds to the W
```

例。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D°M'S.SSS"));
```

```
-2°19'29.928" -3°14'3.243"
```

例。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
```

```
2.3250 degrees S 3.2342 degrees W
```

例。

```
SELECT (ST_AsLatLonText('POINT (-302.2342342 -792.32498)'));
```

```
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='hdddmss');



This function returns a MARC21/XML record with **Coded Cartographic Mathematical Data** representing the bounding box of a given geometry. The format parameter allows to encode the coordinates in subfields \$d,\$e,\$f and \$g in all formats supported by the MARC21/XML standard. Valid formats are:

- cardinal direction, degrees, minutes and seconds (default): hdddmss
- decimal degrees with cardinal direction: hddd.ddddd
- decimal degrees without cardinal direction: ddd.ddddd
- decimal minutes with cardinal direction: hddmm.mmmm
- decimal minutes without cardinal direction: dddmm.mmmm
- decimal seconds with cardinal direction: hddmmss.sss

The decimal sign may be also a comma, e.g. hddmm,mmm.

The precision of decimal formats can be limited by the number of characters after the decimal sign, e.g. hddmm.mm for decimal minutes with a precision of two decimals.

This function ignores the Z and M dimensions.

LOC MARC21/XML versions supported:

- **MARC21/XML 1.1**

Availability: 3.3.0

**Note**

This function does not support non lon/lat geometries, as they are not supported by the MARC21/XML standard (Coded Cartographic Mathematical Data).

**Note**

The MARC21/XML Standard does not provide any means to annotate the spatial reference system for Coded Cartographic Mathematical Data, which means that this information will be lost after conversion to MARC21/XML.



Converting a POINT to MARC21/XML formatted as hdddmss (default)

```
SELECT ST_AsMARC21('SRID=4326;POINT(-4.504289 54.253312)::geometry');

          st_asmarc21
-----
<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" ">
    <subfield code="a"
>a</subfield>
    <subfield code="d"
>W0043015</subfield>
    <subfield code="e"
```

```

>W0043015</subfield>
      <subfield code="f"
>N0541512</subfield>
      <subfield code="g"
>N0541512</subfield>
      </datafield>
    </record>

```

Converting a POLYGON to MARC21/XML formatted in decimal degrees

```

SELECT ST_AsMARC21('SRID=4326;POLYGON((-4.5792388916015625 ↵
54.18172660239091,-4.56756591796875 ↵
54.196993557130355,-4.546623229980469 ↵
54.18313300502024,-4.5792388916015625 54.18172660239091))'::geometry,' ↵
hddd.dddd');

<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" ">
    <subfield code="a"
>a</subfield>
    <subfield code="d"
>W004.5792</subfield>
    <subfield code="e"
>W004.5466</subfield>
    <subfield code="f"
>N054.1970</subfield>
    <subfield code="g"
>N054.1817</subfield>
  </datafield>
</record>

```

Converting a GEOMETRYCOLLECTION to MARC21/XML formatted in decimal minutes. The geometries order in the MARC21/XML output correspond to their order in the collection.

```

SELECT ST_AsMARC21('SRID=4326;GEOMETRYCOLLECTION(POLYGON((13.1 ↵
52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1 ↵
52.38333333333333,13.1 52.65)),POINT(-4.5 54.25))'::geometry,'hdddmm. ↵
mmmm');

                                st_asmarc21
-----
<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" ">
    <subfield code="a"
>a</subfield>
    <subfield code="d"
>E01307.0000</subfield>
    <subfield code="e"
>E01331.0000</subfield>
    <subfield code="f"
>N05240.0000</subfield>
    <subfield code="g"

```

```

>N05224.0000</subfield>
      </datafield>
      <datafield tag="034" ind1="1" ind2=" " >
        <subfield code="a"
>a</subfield>
        <subfield code="d"
>W00430.0000</subfield>
        <subfield code="e"
>W00430.0000</subfield>
        <subfield code="f"
>N05415.0000</subfield>
        <subfield code="g"
>N05415.0000</subfield>
      </datafield>
</record>

```

￼￼

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 function performs geometry simplification appropriate to the integer tile coordinate space. After transforming to tile coordinates it snaps coordinates to the integer grid, removes repeated points, and removes points that lie on straight lines. It does not perform general-purpose, scale-dependent simplification such as **ST_Simplify**. To reduce detail before tile conversion, simplify the geometry in map coordinates before calling **ST_AsMVTGeom**.

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 国国国国国国国国国国国国.



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.

国国

```
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));
```

```
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)
```

国国

[ST_AsMVT](#), [ST_TileEnvelope](#), [PostGIS_Wagyu_Version](#)

7.9.3.10 ST_AsMVT

ST_AsMVT — Aggregate function returning a MVT representation of a set of rows.

Synopsis

```
bytea ST_AsMVT(anyelement set row);
bytea ST_AsMVT(anyelement row, text name);
bytea ST_AsMVT(anyelement row, text name, integer extent);
bytea ST_AsMVT(anyelement row, text name, integer extent, text geom_name);
bytea ST_AsMVT(anyelement row, text name, integer extent, text geom_name, text feature_id_name);
```

☒☒

An aggregate function which returns a binary **Mapbox Vector Tile** representation of a set of rows corresponding to a tile layer. The rows must contain a geometry column which will be encoded as a feature geometry. The geometry must be in tile coordinate space and valid as per the **MVT specification**. **ST_AsMVTGeom** can be used to transform geometry into tile coordinate space. Other row columns are encoded as feature attributes.

The **Mapbox Vector Tile** format can store features with varying sets of attributes. To use this capability supply a JSONB column in the row data containing Json objects one level deep. The keys and values in the JSONB values will be encoded as feature attributes.

Tiles with multiple layers can be created by concatenating multiple calls to this function using `||` or `STRING_AGG`.



Important

Do not call with a `GEOMETRYCOLLECTION` as an element in the row. However you can use **ST_AsMVTGeom** to prepare a geometry collection for inclusion.

row row data with at least a geometry column.

name is the name of the layer. Default is the string "default".

extent is the tile extent in screen space as defined by the specification. Default is 4096.

geom_name is the name of the geometry column in the row data. Default is the first geometry column. Note that PostgreSQL by default automatically **folds unquoted identifiers to lower case**, which means that unless the geometry column is quoted, e.g. `"MyMVTGeom"`, this parameter must be provided as lowercase.

feature_id_name is the name of the Feature ID column in the row data. If NULL or negative the Feature ID is not set. The first column matching name and valid type (smallint, integer, bigint) will be used as Feature ID, and any subsequent column will be added as a property. JSON properties are not supported.

Enhanced: 3.0 - added support for Feature ID.

Enhanced: 2.5.0 - added support parallel query.

2.2.0 ☒☒☒☒☒☒☒☒☒☒.

☒☒

```
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;
```


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);
```

```
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) 格式化为文本。

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) 格式化为文本。TWKB 是一种紧凑的二进制格式，用于存储和传输几何数据。它比 WKB 格式更紧凑，并且支持更多的几何类型。

该函数接受一个几何对象（或几何对象数组）并返回其 TWKB 表示。如果提供了精度参数，则结果将使用指定的精度进行舍入。如果提供了 with_sizes 参数，则结果将包含几何对象的尺寸信息。如果提供了 with_boxes 参数，则结果将包含几何对象的包围盒信息。

该函数还支持将几何对象数组转换为 TWKB 格式。在这种情况下，ids 参数用于指定每个几何对象的 ID。该函数返回一个包含 TWKB 数据和 ID 的数组。

该函数还支持将几何对象转换为 TWKB 格式并返回其文本表示。在这种情况下，array_agg 参数用于指定聚合函数。该函数返回一个包含 TWKB 数据和聚合结果的数组。



Note

<https://github.com/TWKB/Specification> 提供了 TWKB 格式的完整规范。有关 TWKB 格式的实现，请参阅 <https://github.com/TWKB/twkb.js>。

Enhanced: 2.4.0 memory and speed improvements.

2.2.0 支持几何对象数组。

描述

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

```
\x02000202020808
```



```

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

图 6: 由 PostGIS 细分和挤出形成的建筑

```

SELECT ST_AsX3D(ST_Translate(ST_Force3D(ST_Buffer(ST_Point(10,10),5, 'quad_segs=2')), 0,0,
3),
6) As x3dfrag;

```

x3dfrag

```
-----
<IndexedFaceSet  convex='false' 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>
```

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>
```

MULTILINESTRING (MULTILINESTRING)

```
SELECT ST_AsX3D(ST_GeomFromEWKT('MULTILINESTRING((20 0 10,16 -12 10,0 -16 10,-12 -12 10,-20 ←
0 10,-12 16 10,0 24 10,16 16 10,20 0 10),
(12 0 10,8 8 10,0 12 10,-8 8 10,-8 0 10,-8 -4 10,0 -8 10,8 -4 10,12 0 10)))')
) As x3dfrag;
```

x3dfrag

```
-----
<IndexedLineSet  coordIndex='0 1 2 3 4 5 6 7 0 -1 8 9 10 11 12 13 14 15 8'>
  <Coordinate point='20 0 10 16 -12 10 0 -16 10 -12 -12 10 -20 0 10 -12 16 10 0 24 10 16  ←
16 10 12 0 10 8 8 10 0 12 10 -8 8 10 -8 0 10 -8 -4 10 0 -8 10 8 -4 10 ' />
</IndexedLineSet>
```

7.9.3.14 ST_GeoHash

ST_GeoHash — 返回 GeoHash 字符串。

Synopsis

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

国国

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 国国国国国国国国国国国国.



Note

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



This method supports Circular Strings and Curves.

国国

```
SELECT ST_GeoHash(ST_Point(-126,48));
```

```
c0w3hf1s70w3hf1s70w3
```

```
SELECT ST_GeoHash(ST_Point(-126,48), 5);
```

```
c0w3h
```

This example shows that the GeoHash for a line containing a point is a prefix of the point code.

```
SELECT ST_GeoHash('LINESTRING(-126 48, -126.1 48.1)::geometry');
```

```
c0w3
```

国国

ST_GeomFromGeoHash, ST_PointFromGeoHash, ST_Box2dFromGeoHash

7.10 国国国 (operator)

7.10.1 Bounding Box Operators

7.10.1.1 &&

&& — A 国 2D 国国国国国 B 国 2D 国国国国国国国国国国国 TRUE 国国国国国国.

Synopsis

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

&& returns TRUE if the 2D bounding boxes of A and B intersect.



Note

The operand must be a geometry or geography type.

Since 2.0.0 (polyhedral surface) support.
1.5.0 support.

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

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

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

ST_Intersects, **ST_Extent**, **|&>**, **&>**, **&<|**, **&<**, **~**, **@**

7.10.1.2 **&&(geometry,box2df)**

&&(geometry,box2df) — Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

Synopsis

boolean **&&**(geometry A , box2df B);



The `&&` operator returns TRUE if the cached 2D bounding box of geometry A intersects the 2D bounding box B, using float precision. This means that if B is a (double precision) `box2d`, it will be internally converted to a float precision 2D bounding box (BOX2DF)

**Note**

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.



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 INdices (BRIN) was introduced.



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.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

例

```
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)
```

例

`&&(geometry,box2df), &&(box2df,geometry), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @ (geometry,box2df), @ (box2df,geometry), @ (box2df,box2df)`

7.10.1.5 &&&

&&& — A 与 B 不相交返回 TRUE。

Synopsis

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

例

&&& 返回 A 与 B 不相交返回 TRUE。



Note

操作数 (operand) 必须是几何体。

2.0.0 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.

例: 3 维

```
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.

- ✓ 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 INdexes (BRIN) was introduced.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



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



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

￼￼

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

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


Synopsis

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

描述

&< 判断 A 几何体 B 是否完全包含 B 几何体。如果 B 完全包含 A，则返回 TRUE，否则返回 FALSE。



Note

几何体 (operand) 必须是有效的几何体。

示例

```
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)

描述

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

7.10.1.10 &<|

&<| — A 几何体 B 是否完全包含 B 几何体。如果 B 完全包含 A，则返回 TRUE，否则返回 FALSE。

Synopsis

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

描述

&<| 判断 A 几何体 B 是否完全包含 B 几何体。如果 B 完全包含 A，则返回 TRUE，否则返回 FALSE。



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

**Note**

运算符 (operand) 必须都是几何类型。

例

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

column1	column1	overbelow
1	2	f
1	3	t
1	4	t

(3 rows)

例

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

7.10.1.11 &>

&> — A 与 B 不相交返回 TRUE。

Synopsis

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

例

&> 返回 A 与 B 不相交返回 TRUE，相交返回 FALSE。

**Note**

运算符 (operand) 必须都是几何类型。

例

```
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;
```

 $\&\&, |\&\rangle, \&\langle|, \&\langle$

7.10.1.12 <<

```
<< - A B TRUE.
```

Synopsis

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



<< ☐ ☐ ☐ ☐ ☐ ☐ A ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ B ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ TRUE ☐ ☐ ☐ ☐ ☐.



Note

XXXXXXXX (operand) XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.



```
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;
```

```

      1 |      4 | t
(3 rows)

```

☐☐

>>, |>>, <<|

7.10.1.13 <<|

<<| — A 与 B 是否不相交 TRUE 或 FALSE。

Synopsis

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

☐☐

<<| 返回 A 与 B 是否不相交 TRUE 或 FALSE。



Note

☐☐☐☐ (operand) 必须是 geometry 类型。

☐☐

```

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;
```




$\ll, | \gg, \ll |$

7.10.1.16 @

@ — B ☐ ☐ ☐ ☐ ☐ ☐ A ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ TRUE ☐ ☐ ☐ ☐.

Synopsis

```
boolean @( geometry A , geometry B );
```



@ B A TRUE.



Note

[illegible]

```
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;
```


~ , &&

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.



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;
```

```
t
```



**&&(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.



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;
```

t

❏

**&&(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 INdexes (BRIN) was introduced.



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;
```

t

❏

**&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry),
~(box2df,box2df), @(geometry,box2df), @(box2df,geometry)**

7.10.1.20 |&>

|&> — A 与 B 是否相交 TRUE 或 FALSE。

Synopsis

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

描述

|&> 返回 A 与 B 是否相交。如果相交，则返回 TRUE，否则返回 FALSE。

**Note**

操作数 (operand) 必须是几何类型。

描述

```
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)

描述

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

7.10.1.21 |>>

|>> — A 是否完全包含 B 是否 TRUE 或 FALSE。

Synopsis

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



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



Note

XXXXXXXX (operand) XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.



```
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

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.

- ✔ 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;
```

t

☒☒

`&&(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 INdexes (BRIN) was introduced.



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;
```

t

☒☒

`&&(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).

1.5.0

 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](#).

```
SELECT 'LINESTRING(0 0, 1 1)::geometry ~= 'LINESTRING(0 1, 1 0)::geometry AS equality;

t
```

[ST_Equals](#), [ST_OrderingEquals](#), =

7.10.2 (operator)

7.10.2.1 <->

<-> — A B 2

Synopsis

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

The <-> operator returns the 2D distance between two geometries. Used in the "ORDER BY" clause provides index-assisted nearest-neighbor result sets with true KNN distance search for geometries, and distance on the sphere for geographies.



Note

(operand) 2 GiST ORDER BY



Note

a.geom 'SRID=3005;POINT(1011102 450541)::geometry

文档 OpenGeo workshop: Nearest-Neighbour Searching 文档.

Enhanced: 2.2.0 -- True KNN ("K nearest neighbor") behavior for geometry and geography. Note for geography KNN is based on sphere rather than spheroid.

Changed: 2.2.0 -- Old hybrid-syntax workarounds may be slower once true KNN is available. See examples below.

Availability: 2.0.0 -- Weak KNN provides nearest neighbors based on geometry centroid distances instead of true distances. Exact results for points, inexact for all other types.

文档

```
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)

文档 KNN 文档:

```
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)

文档 "EXPLAIN ANALYZE" 文档.

When a plan cannot produce exact distances directly, use a hybrid query to find the true nearest neighbors. First a CTE query using the index-assisted KNN, then an exact query to get correct ordering:

```
WITH index_query AS (
  SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)::geometry') as d,edabbr, vaabbr
  FROM va2005
```


SQL

```
SELECT *
FROM (
SELECT b.tlid, b.mtfcc,
       b.geom <#
> ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
                             745787 2948499,745740 2948468,745712 2948438,
                             745690 2948384,745677 2948319)',2249) As b_dist,
       ST_Distance(b.geom, ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
                             745787 2948499,745740 2948468,745712 2948438,
                             745690 2948384,745677 2948319)',2249)) As act_dist
FROM bos_roads As b
ORDER BY b_dist, b.tlid
LIMIT 100) As foo
ORDER BY act_dist, tlid LIMIT 10;
```

tlid	mtfcc	b_dist	act_dist
85732027	S1400	0	0
85732029	S1400	0	0
85732031	S1400	0	0
85734335	S1400	0	0
85736037	S1400	0	0
624683742	S1400	0	128.528874268666
85719343	S1400	260.839270432962	260.839270432962
85741826	S1400	164.759294123275	260.839270432962
85732032	S1400	277.75	311.830282365264
85735592	S1400	222.25	311.830282365264

(10 rows)

SQL

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);

SQL

<-> `geometry` `N` `(geometry)` `geometry`. `geometry` `geometry`.



Note

`geometry` (operand) `N` `GiST` `geometry`. `ORDER BY` `geometry` `geometry`.

**Note**

ST_3DIntersects, ST_3DDWithin a.geom ST_3DIntersects SRID=3005;POINT(1011102 450541)::geometry 交叉, (ST_3DIntersects/CTE(common table expression) 交叉) 交叉。交叉。

Availability: 2.2.0.

交叉

<->

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);

交叉

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 交叉。



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

例

```
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);
```

```
t
```

例

[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);

例

Returns TRUE if geometry A contains geometry B. A contains B if and only if all points of B lie inside (i.e. in the interior or boundary of) A (or equivalently, no points of B lie in the exterior of A), and the interiors of A and B have at least one point in common.

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

The contains relationship is reflexive: every geometry contains itself. (In contrast, in the [ST_ContainsProperly](#) predicate a geometry does *not* properly contain itself.) The relationship is antisymmetric: if $ST\_Contains(A, B) = \text{true}$ and $ST\_Contains(B, A) = \text{true}$, then the two geometries must be topologically equal ($ST\_Equals(A, B) = \text{true}$).

ST_Contains is the converse of [ST_Within](#). So, $ST\_Contains(A, B) = ST\_Within(B, A)$.



Note

Because the interiors must have a common point, a subtlety of the definition is that polygons and lines do *not* contain lines and points lying fully in their boundary. For further details see [Subtleties of OGC Covers, Contains, Within](#). The [ST_Covers](#) predicate provides a more inclusive relationship.

**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Contains`.

GEOS 3.10.0

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.



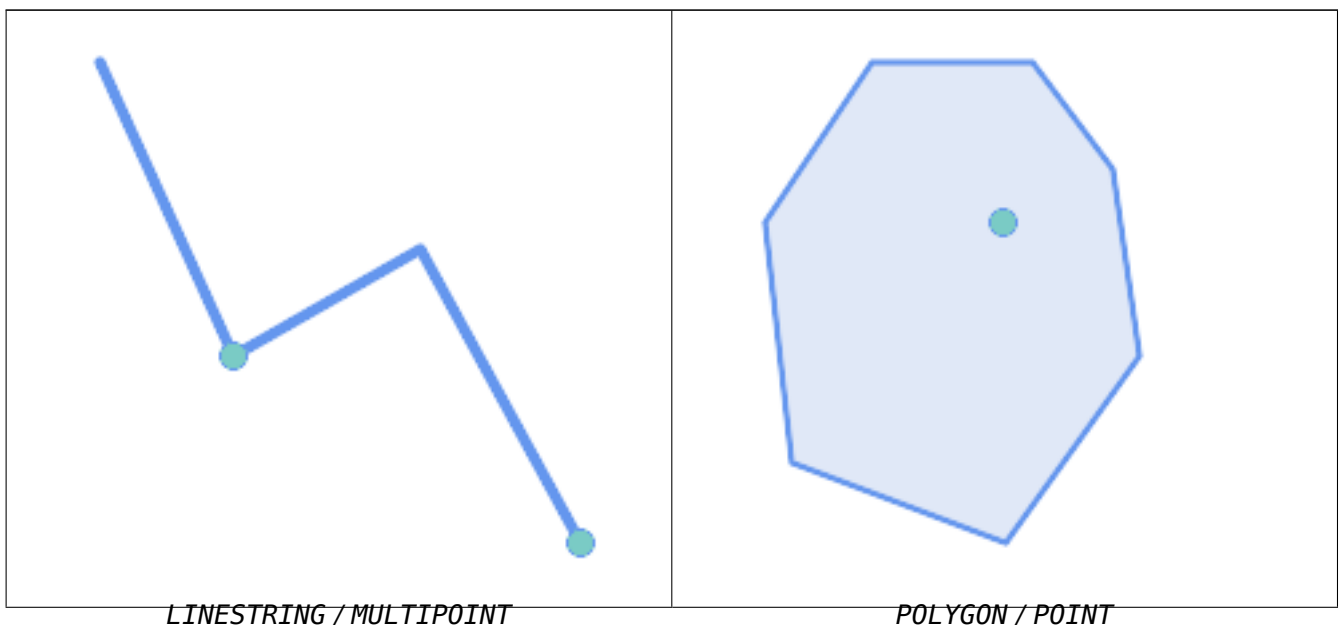
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3](#) - same as `within(geometry B, geometry A)`

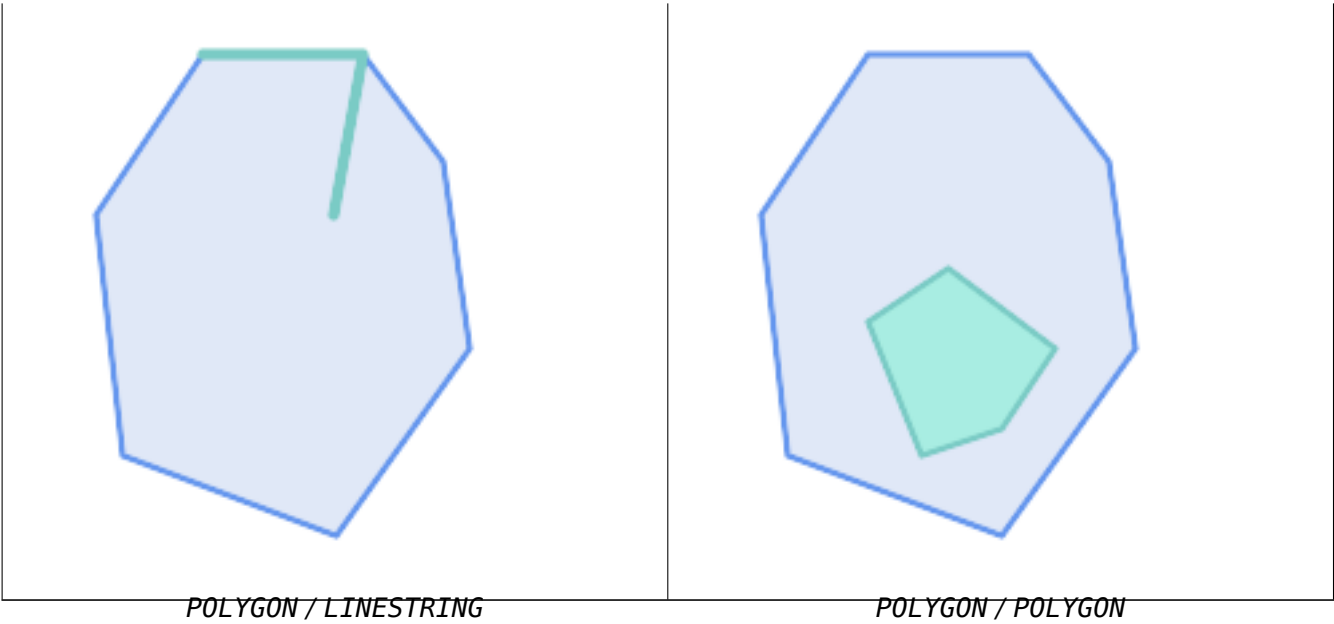


This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

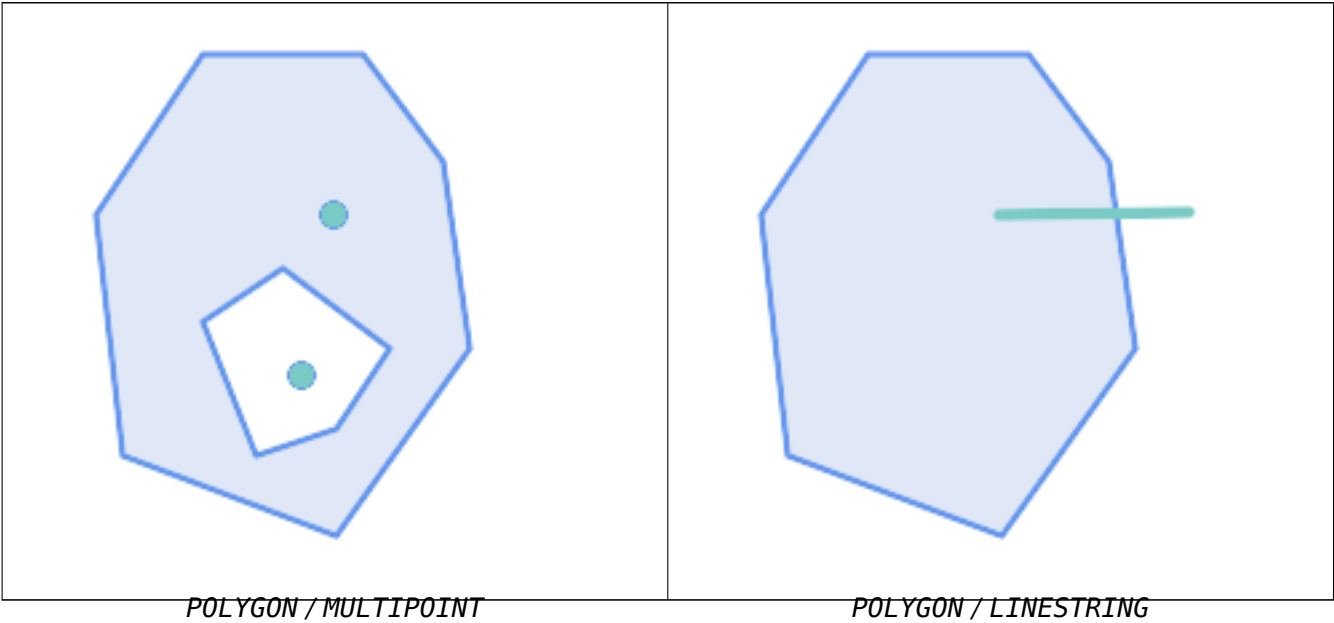
图例

`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):



This example tests a circle within another 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;
```

smallcontainsbig	bigcontainssmall	bigcontainsunion	bigisunion	bigcoversexterior	bigcontainsexterior	
f	t	t	t	t	f	

This example demonstrates the difference between contains and contains-properly predicates.

```
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

测试

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);

国国

Returns true if every point of B lies in the interior of A (or equivalently, no point of B lies in the the boundary or exterior of A).

In mathematical terms: $ST\_ContainsProperly(A, B) \Leftrightarrow Int(A) \cap B = B$

A contains B properly if the DE-9IM Intersection Matrix for the two geometries matches [T**FF*FF*]

A does not properly contain itself, but does contain itself.

A use for this predicate is computing the intersections of a set of geometries with a large polygonal geometry. Since intersection is a fairly slow operation, it can be more efficient to use containsProperly to filter out test geometries which lie fully inside the area. In these cases the intersection is known a priori to be exactly the original test geometry.



Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_ContainsProperly`.



Note

The advantage of this predicate over `ST_Contains` and `ST_Intersects` is that it can be computed more efficiently, with no need to compute topology at individual points.

GEOS 国国国国国

1.4.0 国国国国国国国国国国国.



Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



Important

Do not use this function with invalid geometries. You will get unexpected results.

☒☒

This example tests a circle within another 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;
```

smallcontainspropbig	bigcontainspropsmall	bigcontainspropunion	bigisunion	↔
bigcoversexterior bigcontainsexterior				
-----+-----+-----+-----+-----↔				
f	t	f	t	t ↔
	f			

This example demonstrates the difference between contains and contains-properly predicates.

```
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: $ST\_CoveredBy(A, B) \Leftrightarrow A \sqsubset B = A$

`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.

This example tests a circle covered by another 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;
```

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

This example tests a circle covering another 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;
```

smallinsmall	smallcoversbig	bigcoversexterior	bigcontainsexterior
t	f	t	f
(1 row)			

Geeography Example

This example compares a point with a 300-meter buffer against a point and against a 10-meter buffer around that point.

```
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



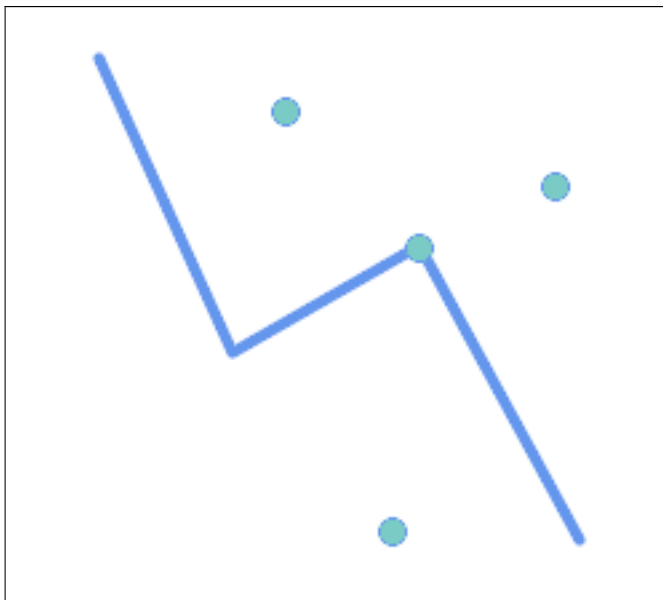
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.13.3](#)



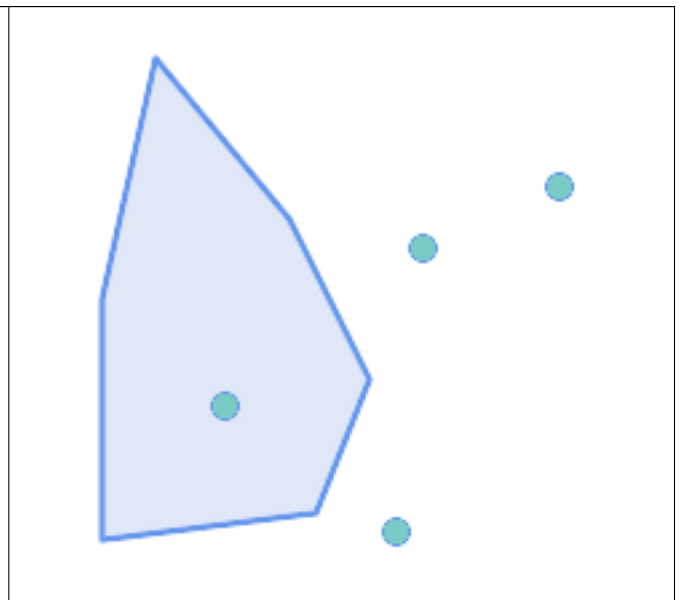
This method implements the SQL/MM specification. SQL-MM 3: 5.1.29



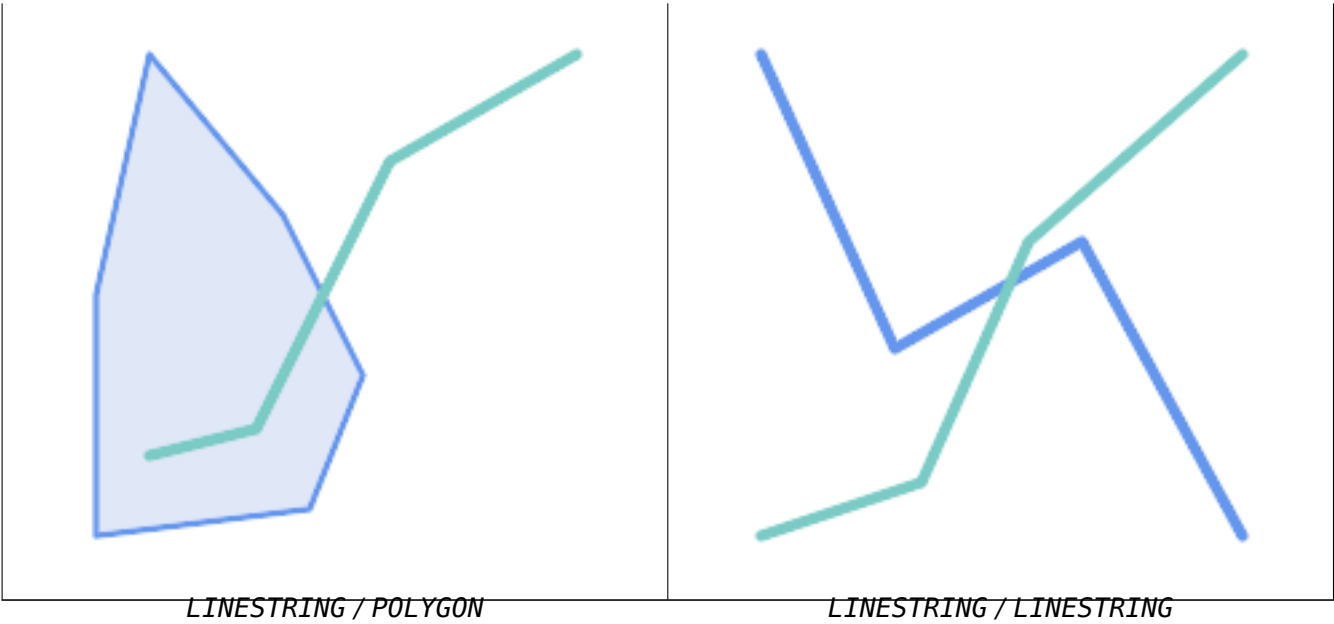
The following situations all return true.



MULTIPOINT / LINESTRING



MULTIPOINT / POLYGON



Consider a situation where a user has two tables: a table of roads and a table of highways.

<pre>CREATE TABLE roads (id serial NOT NULL, geom geometry, CONSTRAINT roads_pkey PRIMARY KEY (↵ road_id));</pre>	<pre>CREATE TABLE highways (id serial NOT NULL, the_gem geometry, CONSTRAINT roads_pkey PRIMARY KEY (↵ road_id));</pre>

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);
```

t

```
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
```

f

￼￼

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)'));
```

t

```
SELECT ST_Equals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)')),
  ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
```

t



[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);
```

f

```
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
```

t

This example looks up rows in a table; for faster lookup, the geometry column should have a GiST index.

```
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))'::geometry);
```

```
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');

```

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:

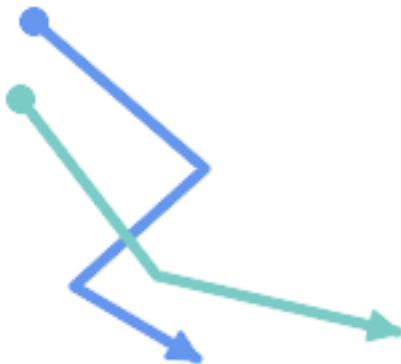
- 0: LINE NO CROSS
- -1: LINE CROSS LEFT

- 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

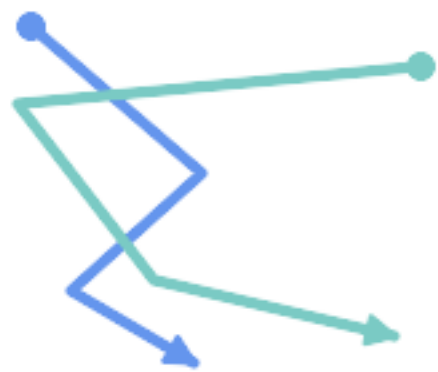


Blue: Line A; Green: Line B

```
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

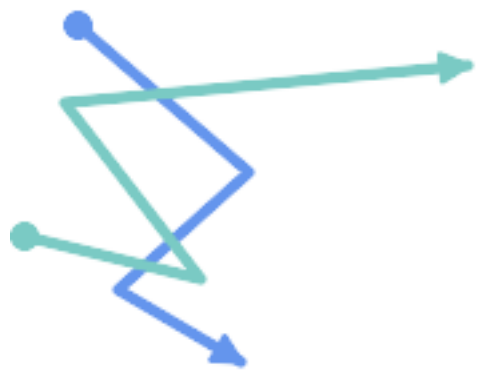


Blue: Line A; Green: Line B

```
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/arcscde/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)');
```

f

```
SELECT ST_OrderingEquals('LINESTRING(0 0, 10 10)',
  'LINESTRING(0 0, 10 10)');
```

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))');
```

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: $ST\_Overlaps(A, B) \Leftrightarrow (dim(A) = dim(B) = dim(Int(A) \cap Int(B))) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$



Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Overlaps`.

GEOS 3.10.0



Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

NOTE: this is the “allowable” version that returns a boolean, not an integer.



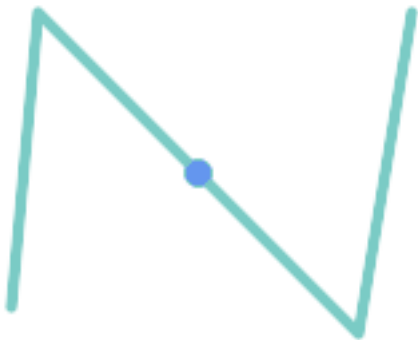
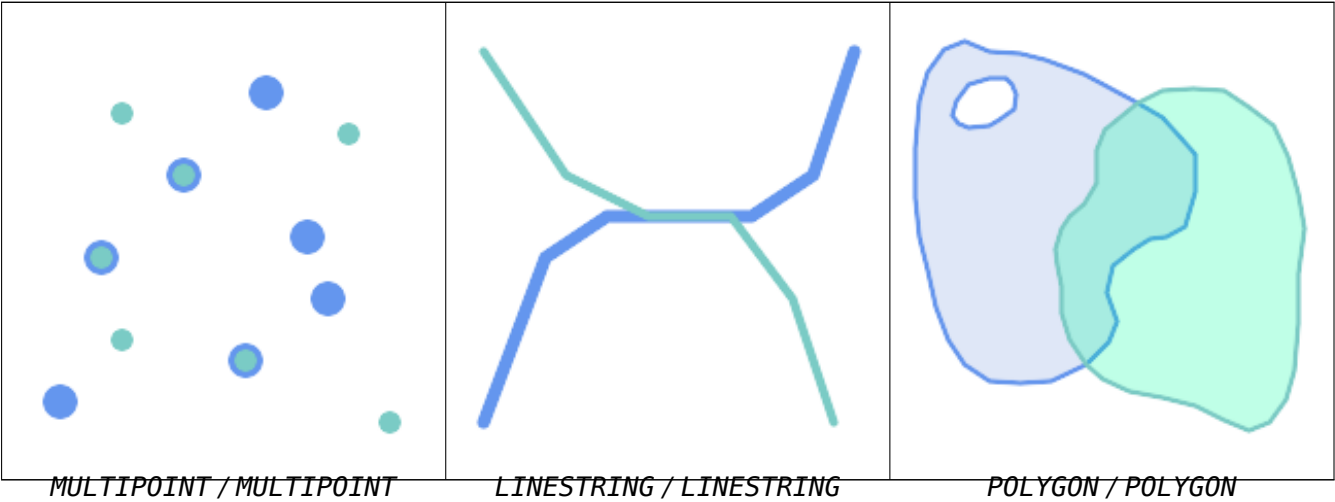
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3](#)



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](#).

Variante 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');
```

```
t
```

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer('POINT(1 2)', 2), '*FF*FF212');
```

```
t
```

Testing a custom spatial relationship pattern as a query condition, with && to enable using a spatial index.

Find compounds that properly intersect (not just touch) a poly (Interior Intersects)

```
SELECT c.* , p.name As poly_name
FROM polys AS p
INNER JOIN compounds As c
ON c.geom && p.geom
AND ST_Relate(p.geom, c.geom, 'T*****');
```

Computing the intersection matrix for spatial relationships.

```
SELECT ST_Relate('POINT(1 2)',
                ST_Buffer('POINT(1 2)', 2));
```

```
0FFFFFF212
```

```
SELECT ST_Relate('LINESTRING(1 2, 3 4)',
                'LINESTRING(5 6, 7 8)' );
```

```
FF1FF0102
```

Using different Boundary Node Rules to compute the spatial relationship between a LineString and a MultiLineString with a duplicate endpoint (3 3):

- Using the **OGC-Mod2** rule (1) the duplicate endpoint is in the **interior** of the MultiLineString, so the DE-9IM matrix entry [aB:bI] is 0 and [aB:bB] is F.
- Using the **Endpoint** rule (2) the duplicate endpoint is in the **boundary** of the MultiLineString, so the DE-9IM matrix entry [aB:bI] is F and [aB:bB] is 0.

```
WITH data AS (SELECT
  'LINESTRING(1 1, 3 3)::geometry AS a_line,
  'MULTILINESTRING((3 3, 3 5), (3 3, 5 3)):: geometry AS b_multiline
)
SELECT ST_Relate(a_line, b_multiline, 1) AS bnr_mod2,
       ST_Relate(a_line, b_multiline, 2) AS bnr_endpoint
FROM data;
```

```
bnr_mod2 | bnr_endpoint
-----+-----
FF10F0102 | FF1F00102
```

☒☒

Section 5.1, [ST_RelateMatch](#), [ST_Contains](#), [ST_ContainsProperly](#), [ST_Covers](#), [ST_CoveredBy](#), [ST_Crosses](#), [ST_Disjoint](#), [ST_Equals](#), [ST_Intersects](#), [ST_Overlaps](#), [ST_Touches](#), [ST_Within](#)

7.11.1.14 ST_RelateMatch

`ST_RelateMatch` — Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern

Synopsis

boolean **ST_RelateMatch**(text intersectionMatrix, text intersectionMatrixPattern);

☒☒

Tests if a **Dimensionally Extended 9-Intersection Model** (DE-9IM) intersectionMatrix value satisfies an intersectionMatrixPattern. Intersection matrix values can be computed by [ST_Relate](#).

For more information refer to Section 5.1.

GEOS ☒☒☒☒☒

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

```
SELECT ST_RelateMatch('101202FFF', 'TTTTTTFFF') ;
```

t

Patterns for common spatial relationships matched against intersection matrix values, for a line in various positions relative to a polygon

```
SELECT pat.name AS relationship, pat.val AS pattern,
       mat.name AS position, mat.val AS matrix,
       ST_RelateMatch(mat.val, pat.val) AS match
FROM (VALUES ( 'Equality', 'T1FF1FFF1' ),
            ( 'Overlaps', 'T*T***T**' ),
            ( 'Within', 'T*F**F***' ),
            ( 'Disjoint', 'FF*FF*****' )) AS pat(name,val)
CROSS JOIN
  (VALUES ('non-intersecting', 'FF1FF0212'),
          ('overlapping', '1010F0212'),
          ('inside', '1FF0FF212')) AS mat(name,val);
```

relationship	pattern	position	matrix	match
Equality	T1FF1FFF1	non-intersecting	FF1FF0212	f
Equality	T1FF1FFF1	overlapping	1010F0212	f
Equality	T1FF1FFF1	inside	1FF0FF212	f
Overlaps	T*T***T**	non-intersecting	FF1FF0212	f
Overlaps	T*T***T**	overlapping	1010F0212	t
Overlaps	T*T***T**	inside	1FF0FF212	f
Within	T*F**F***	non-intersecting	FF1FF0212	f
Within	T*F**F***	overlapping	1010F0212	f
Within	T*F**F***	inside	1FF0FF212	t
Disjoint	FF*FF*****	non-intersecting	FF1FF0212	t
Disjoint	FF*FF*****	overlapping	1010F0212	f
Disjoint	FF*FF*****	inside	1FF0FF212	f

Section 5.1, [ST_Relate](#)

7.11.1.15 ST_Touches

ST_Touches — Tests if two geometries have at least one point in common, but their interiors do not intersect

Synopsis

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

Returns TRUE if A and B intersect, but their interiors do not intersect. Equivalently, A and B have at least one point in common, and the common points lie in at least one boundary. For Point/Point inputs the relationship is always FALSE, since points do not have a boundary.

In mathematical terms: $ST\_Touches(A, B) \Leftrightarrow (Int(A) \cap Int(B) = \emptyset) \wedge (A \cap B \neq \emptyset)$

This relationship holds if the DE-9IM Intersection Matrix for the two geometries matches one of:

- FT*****
- F**T*****
- F***T****



Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid using an index, use `_ST_Touches` instead.



Important

Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`



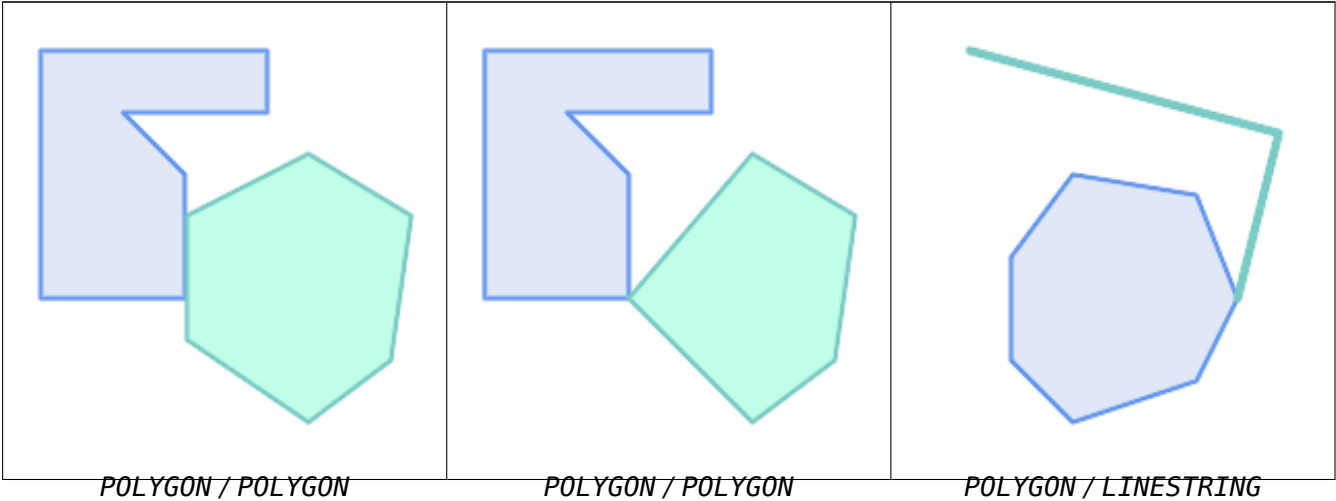
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3](#)

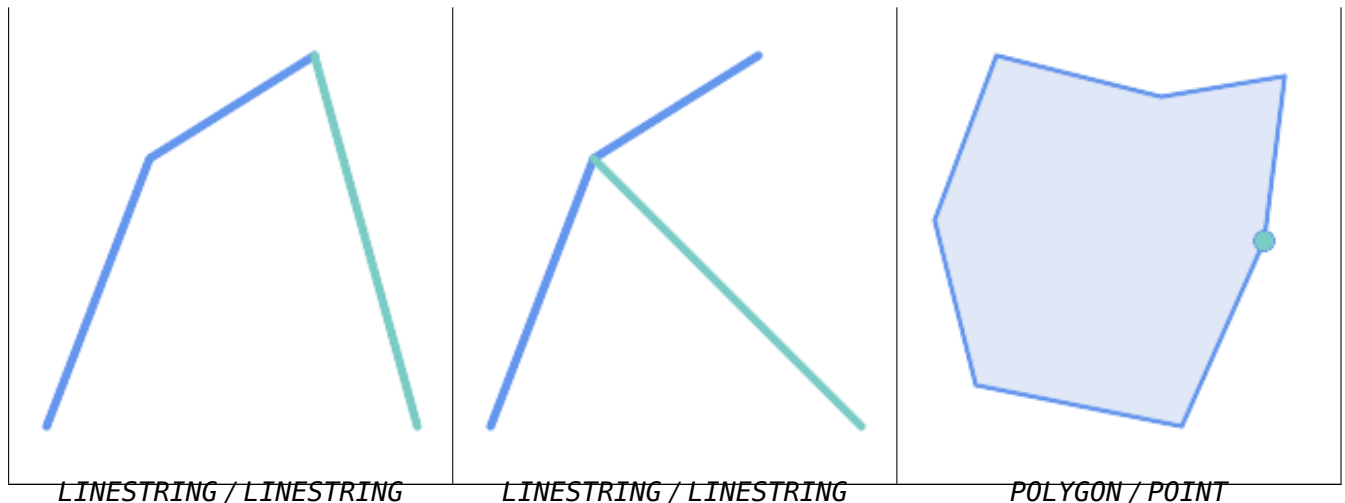


This method implements the SQL/MM specification. SQL-MM 3: 5.1.28



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





```
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(1 1)::geometry');
```

```
f
```

```
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(0 2)::geometry');
```

```
t
```

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

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

This example tests a circle within another 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;
```

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 that 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 a 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

例

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

```
t
```

例

ST_DWithin

7.12 Measurement Functions

7.12.1 ST_Area

ST_Area — 返回几何体的面积。

Synopsis

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

例

ST_Surface 和 ST_MultiSurface 返回的几何体 - 曲面。SRID 2 表示曲面。ST_Area(geog,true) 返回曲面 (curved surface) 的面积。ST_Area(geog,false) 返回平面面积。

例: 2.0.0 版本 2 表示多面体 (polyhedral surface) 的面积。

例: 2.2.0 版本使用 GeographicLib 库。Proj 4.9.0 版本使用 Proj 4.9.0 库。

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

ST_Area(geog, (2.5 表示曲面) 2 表示多面体)。2.5 表示 0 表示 (non-zero) 表示 XY 平面面积。

 (plot) . EPSG:2249 .

Return area square feet and transform to Massachusetts state plane meters (EPSG:26986) to get square meters. Note this is in square feet because 2249 is Massachusetts State Plane Feet and transformed area is in square meters since EPSG:26986 is state plane Massachusetts meters.

[illegible]

```

select ST_Area(geog) / 0.3048 ^ 2 sqft_spheroid,
       ST_Area(geog, false) / 0.3048 ^ 2 sqft_sphere,
       ST_Area(geog) sqm_spheroid
from (
    select ST_Transform(
        'SRID=2249;POLYGON((743238 2967416,743238 2967450,743265
        2967450,743265.625 2967416,743238 2967416))':::geometry,
        4326
    ) :: geography geog
    ) as subquery;

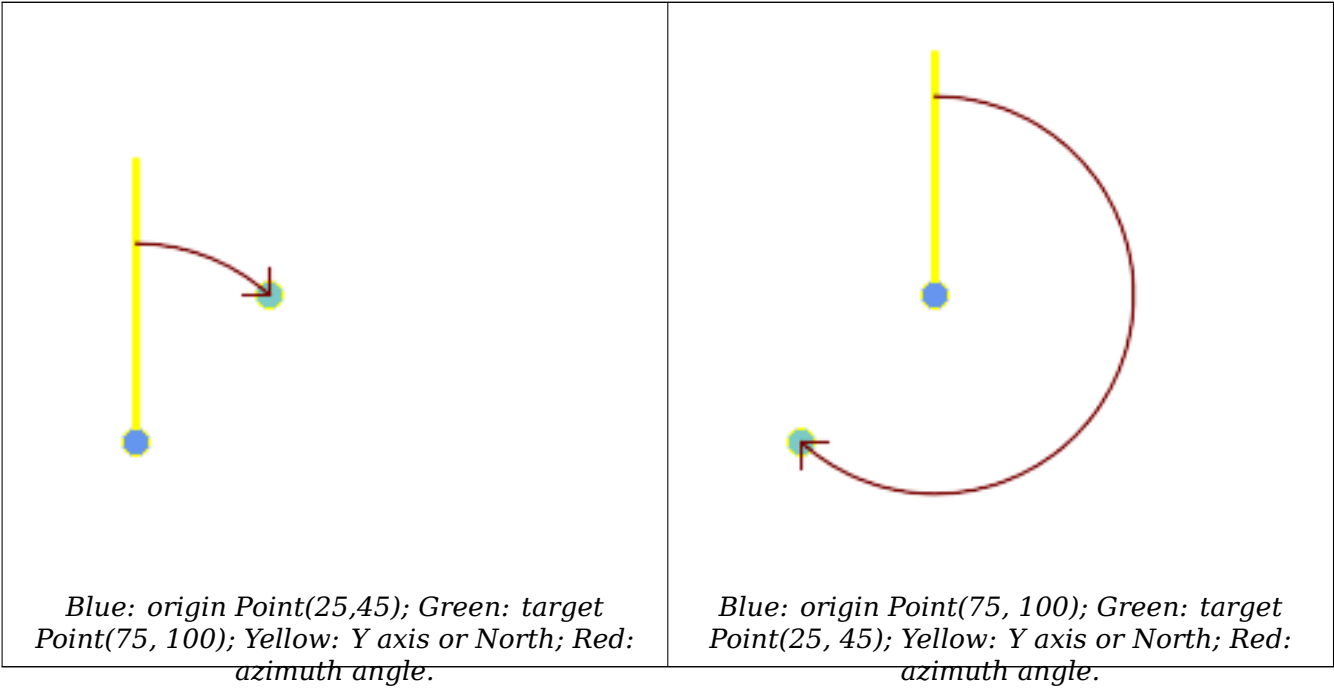
```


图例

图例图例图例图例

```
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
-----+	-----
42.27368900609371	222.27368900609372



图例

ST_Angle, ST_Translate, ST_Project, PostgreSQL Math Functions

7.12.3 ST_Angle

ST_Angle — 返回 3 个点 (最长) 所形成的角度。

Synopsis

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

图例

图例图例图例 3 图例图例 (longest) 图例图例图例图例。

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 `degrees()` `ST_Angle()`. `ST_Angle()` `degrees()`.

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

Availability: 2.5.0

```
SELECT degrees(ST_Angle('POINT(0 0)', 'POINT(10 10)', 'POINT(20 0)'));
```

270

Angle between vectors defined by four points

```
SELECT degrees(ST_Angle('POINT (10 10)', 'POINT (0 0)', 'POINT(90 90)', 'POINT (100 80)'));
```

269.99999999999999

Angle between vectors defined by the start and end points of lines

```
SELECT degrees(ST_Angle('LINESTRING(0 0, 0.3 0.7, 1 1)', 'LINESTRING(0 0, 0.2 0.5, 1 0)'));
```

45

ST_Azimuth

7.12.4 ST_ClosestPoint

ST_ClosestPoint — Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.

Synopsis

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

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



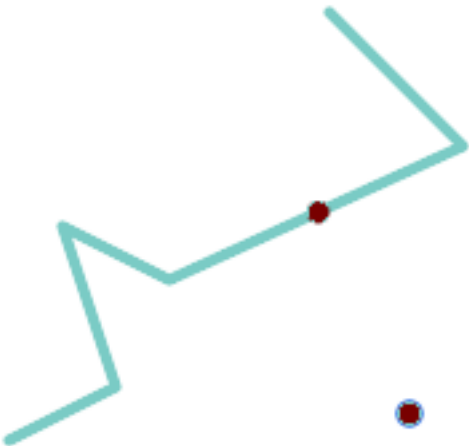
Note

3 **ST_3DClosestPoint**

Enhanced: 3.4.0 - Support for geography.

1.5.0

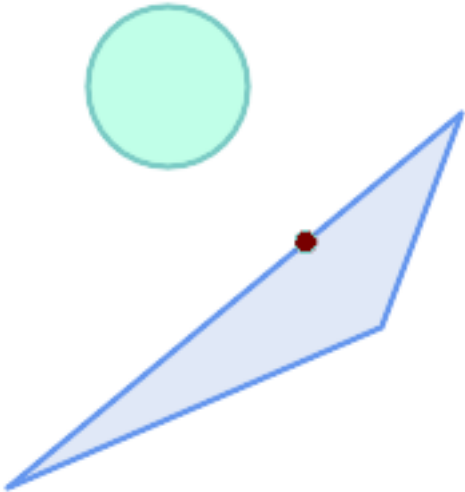
點



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 101.89887534906195)
```


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(54.69937988676193 128.93502291722837 11.547586950660556) | POINT(50 75)
```

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(40.45454545454545 53.63636363636364 5) | POINT(20 40)
```

ST

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);
```

ST

ST_Distance, ST_3DDistance 3D 最长距离 (longest) 最近点 (SRS 无关) 最近点。

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 最近点。ST_Distance 3D 最长距离 (longest) 最近点。ST_Distance 3D 最长距离 (longest) 最近点。

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

修复: 2.1.0 修复了某些地理数据类型的问题。

修复: 2.2.0 修复了某些地理数据类型的问题，GeographicLib 修复。修复了 Proj 4.9.0 修复了某些地理数据类型的问题。

Changed: 3.0.0 - does not depend on SFCGAL anymore.

修复

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

文档

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

文档

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

7.12.7 ST_3DDistance

ST_3DDistance — 计算两个几何体 (SRID 3D) 之间的 3D 距离。

Synopsis

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

文档

计算两个几何体 (SRID 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 到 3D 文档 Z 文档 Z 到 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 that 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 | 0
```

☒☒

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

7.12.8 ST_DistanceSphere

`ST_DistanceSphere` — 计算两个几何对象在球面上的距离。PostGIS 1.5 引入。

Synopsis

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

☒☒

返回两个几何对象在球面上的距离。SRID 必须为 4326。PostGIS 1.5 引入。

1.5 版本中，`ST_DistanceSphere` 函数返回的距离值可能为负数。

☒☒☒: 2.2.0 版本中，`ST_Distance_Sphere` 函数被引入。

SQL

```
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
70424.71	70438.00	0.72900	65871.18

SQL

ST_Distance, **ST_DistanceSpheroid**

7.12.9 ST_DistanceSpheroid

ST_DistanceSpheroid — Computes the distance between two geometries using spherical Earth model taking into account the curvature of the Earth. PostGIS 1.5 introduced this function.

Synopsis

float **ST_DistanceSpheroid**(geometry geom1lonlatA, geometry geom1lonlatB, spheroid measurement_spheroid)

SQL

Computes the distance between two geometries using spherical Earth model taking into account the curvature of the Earth. PostGIS 1.5 introduced this function. **ST_LengthSpheroid**



Note

SRID must be specified for both geometries. If not specified, the SRID of the first geometry is used.

1.5 introduced this function. 1.5 introduced this function.

2.2.0 introduced ST_Distance_Spheroid.

例

```
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	70424.71	70438.00

例

ST_Distance, ST_DistanceSphere

7.12.10 ST_FrechetDistance

ST_FrechetDistance — 计算两个几何体之间的最短距离 (shortest)。

Synopsis

float **ST_FrechetDistance**(geometry g1, geometry g2, float densifyFrac = -1);

例

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

ST_FrechetDistance 函数在计算两个几何体之间的最短距离时，会先对每个几何体进行分段稠密化。分段稠密化参数 densifyFrac 设置了对每个分段进行稠密化的比例。每个分段将被分割成若干个等长度的子分段，其长度之和最接近于给定的比例。

**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 3.10.0

Availability: 2.4.0 - requires GEOS >= 3.7.0

SQL

```
SELECT ST_FrechetDistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, 50 50, 100 50)::geometry');
```

```
70.7106781186548
```

```
SELECT ST_FrechetDistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, 50 50, 100 50)::geometry, 0.5);
```

```
50
```

SQL

ST_HausdorffDistance

7.12.11 ST_HausdorffDistance

`ST_HausdorffDistance` — 返回两个几何体的最短距离 (shortest)。

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 简体中文.

图



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);
```

图

ST_FrechetDistance

7.12.12 ST_Length

ST_Length — 返回几何体的长度。

Synopsis

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

几何体

几何体: 点, 线, ST_Curve, ST_MultiCurve 返回 2 维几何体的长度。返回 0 表示空几何体。对于 geography 类型, **ST_Perimeter** 返回周长。对于 geography 类型, 返回周长。

For geography types: computation is performed using the inverse geodesic calculation. Units of length are in meters. The spheroid is specified by the SRID; when no SRID is provided, WGS84 is used. If use_spheroid = false, then the calculation is based on a sphere instead of a spheroid.

对于 geography 类型, ST_Length2D 返回长度, 使用平面几何计算。



Warning

警告: 2.0.0 版本中, ST_Length 对于 geography 类型返回的是平面长度。2.0.0 版本中, ST_Length 对于 geography 类型返回的是平面长度。2.0.0 版本中, ST_Length 对于 geography 类型返回的是平面长度。2.0.0 版本中, ST_Length 对于 geography 类型返回的是平面长度。



Note

对于 geography 类型, ST_Length 返回的是平面长度。对于 geography 类型, ST_Length 返回的是平面长度。对于 geography 类型, ST_Length 返回的是平面长度。对于 geography 类型, ST_Length 返回的是平面长度。



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 版本中, ST_Length 返回的是平面长度。

几何体

几何体: 点, 线, ST_Curve, ST_MultiCurve 返回 2 维几何体的长度。返回 0 表示空几何体。对于 geography 类型, **ST_Perimeter** 返回周长。对于 geography 类型, 返回周长。

```
SELECT ST_Length(ST_GeomFromText(  
'LINESTRING(743238 2967416,743238 2967450,743265 2967450,  
743265.625 2967416,743238 2967416)',  
2249));
```

122.63074400009504

This example transforms a 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
));
```

```
34309.4563576191
```

注意

WGS84 坐标系。

This example uses the default spheroid calculation.

```
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
-----+-----
34310.5703627284 | 34346.20609607416
```

注意

[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。返回类型是 `float`。

Synopsis

```
float ST_Length2D(geometry a_2dlinestring);
```

注意

返回类型是 `float`。返回类型是 `float`。返回类型是 `float`。

注意

[ST_Length](#), [ST_3DLength](#)

7.12.14 ST_3DLength

`ST_3DLength` — 返回 3D 空间中的长度。

Synopsis

float **ST_3DLength**(geometry a_3dlinestring);

返回

返回浮点型值，表示 3D 线段的长度。2D 线段的长度由 ST_Length 函数返回。2D 线段的长度由 ST_Length2D 函数返回。



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



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 7.1, 10.3

版本: 2.0.0 在 ST_Length3D 函数中实现。

返回

3D 线段的长度。使用 EPSG:2249 坐标系。返回浮点型值。

```
SELECT ST_3DLength(ST_GeomFromText(
'LINESTRING(743238 2967416 1,743238 2967450 1,743265 2967450 3,
743265.625 2967416 3,743238 2967416 3)',
2249));
```

```
122.70471674145682
```

返回

ST_Length, ST_Length2D

7.12.15 ST_LengthSpheroid

ST_LengthSpheroid — 返回浮点型值。

Synopsis

float **ST_LengthSpheroid**(geometry a_geometry, spheroid a_spheroid);

返回

Calculates the length or perimeter of a geometry on a spheroid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection. The spheroid is specified by a text value as follows:

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

返回

```
SPHEROID["GRS_1980",6378137,298.257222101]
```

1.2.2 使用 ST_LengthSpheroid 函数。

PostGIS 2.2.0 版本开始支持 ST_LengthSpheroid 函数，ST_3DLengthSpheroid 函数在 2.2.0 版本中尚未实现。

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

SQL

```
SELECT ST_LengthSpheroid(
    geometry_column,
    'SPHEROID["GRS_1980",6378137,298.257222101]'
)
FROM geometry_table;
```

```
SELECT ST_LengthSpheroid(geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(geom,2), sph_m) As len_line2
FROM (SELECT ST_GeomFromText('MULTILINESTRING((-118.584 38.374,-118.583 38.5),
(-71.05957 42.3589 , -71.061 43))') As geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m) as foo;
```

tot_len	len_line1	len_line2
85204.52077117894	13986.87252824332	71217.64824293563

This example uses 3D input.

```
SELECT ST_LengthSpheroid(geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(geom,2), sph_m) As len_line2
FROM (SELECT ST_GeomFromEWKT('MULTILINESTRING((-118.584 38.374 20,-118.583 38.5 30),
(-71.05957 42.3589 75, -71.061 43 90))') As geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m) as foo;
```

tot_len	len_line1	len_line2
85204.52592562366	13986.876103023424	71217.64982260024

SQL

ST_GeometryN, ST_Length

7.12.16 ST_LongestLine

ST_LongestLine — 返回 3 个最长的线段的几何对象。

Synopsis

```
geometry ST_LongestLine(geometry g);
geometry ST_LongestLine(geometry g1, geometry g2);
```

描述

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

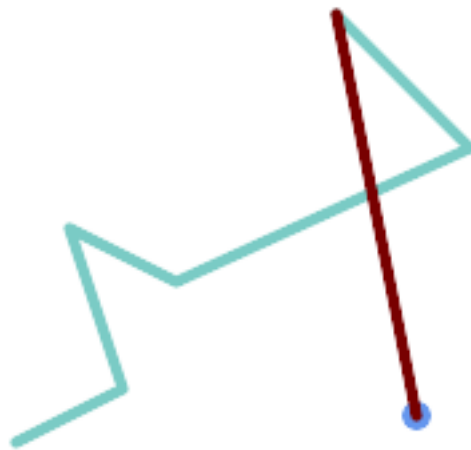
The longest line always occurs between two vertices. The function returns the first longest line if more than one is found. The length of the line is equal to the distance returned by [ST_MaxDistance](#).

If only one geometry is provided, or 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 简体中文版。

Enhanced: 3.7.0 - support for a single geometry input.

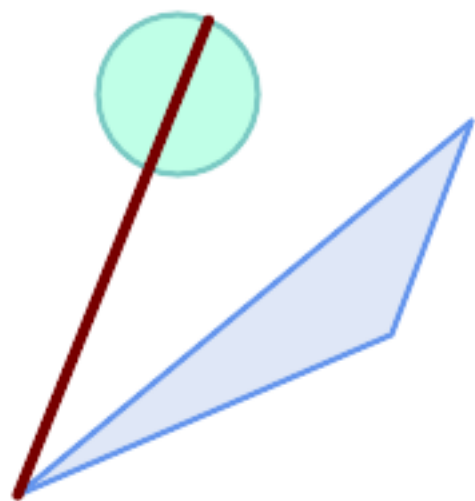
示例



SQL 函数调用

```
SELECT ST_AsText(ST_LongestLine(
    'POINT (160 40)',
    'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)' )
    ) AS lline;
```

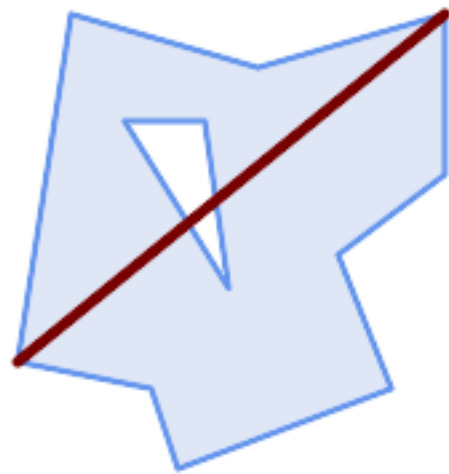
```
LINESTRING(160 40,130 190)
```



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 10,91.4805029709527 187.7163859753386)
```



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)) AS llinewkt,
       ST_MaxDistance(geom) AS max_dist,
       ST_Length(ST_LongestLine(geom)) AS lenll
FROM (SELECT 'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 ←
50, 40 180),
       (60 140, 99 77.5, 90 140, 60 140))'::geometry AS geom) AS t;
```

llinewkt		max_dist		lenll
-----+-----+-----				
LINESTRING(20 50,180 180)		206.15528128088303		206.15528128088303



ST_MaxDistance, ST_ShortestLine, ST_3DLongestLine, ST_MinimumBoundingCircle

7.12.17 ST_3DLongestLine

ST_3DLongestLine — 3D longest line.

Synopsis

```
geometry ST_3DLongestLine(geometry g1, geometry g2);
```


3 (longest) . ,
 . g1 g2 . 3
ST_3DMaxDistance g1 g2 .

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

Figure 2.2.0: 2D plot of $\rho(x, y, z)$, ($\rho(x, y, z) \geq 0$ for all x, y, z) 2D plot of $\rho(x, y, z)$. 2D plot of $\rho(x, y, z)$, ($\rho(x, y, z) \geq 0$ for all x, y, z).

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

[illegible]

```
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
       ST_AsEWKT(ST_LongestLine(line,pt)) AS lol2d_line_pt
FROM (SELECT 'POINT(100 100 30) '::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000) '::
geometry As line
      ) As foo;
```

```

lol3d_line_pt      |      lol2d_line_pt
-----+-----
LINESTRING(50 75 1000,100 100 30) | LINESTRING(98 190,100 100)

```

☒☒☒☒☒☒☒☒☒☒ -- 3D, 2D ☒☒☒☒☒☒☒☒☒☒

```
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 几何类型

```
SELECT ST_AsEWKT(ST_3DLongestLine(poly, mline)) As lol3d,
       ST_AsEWKT(ST_LongestLine(poly, mline)) As lol2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,
100 100 5, 175 150 5))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125
100 1, 175 155 1),
       (1 10 2, 5 20 1))') As mline ) As foo;

lol3d          |          lol2d
-----+-----
LINESTRING(175 150 5,1 10 2) | LINESTRING(175 150,1 10)
```

3D

[ST_3DClosestPoint](#), [ST_3DDistance](#), [ST_LongestLine](#), [ST_3DShortestLine](#), [ST_3DMaxDistance](#)

7.12.18 ST_MaxDistance

`ST_MaxDistance` — 返回 2 个几何体之间的最大距离。

Synopsis

```
float ST_MaxDistance(geometry g);
float ST_MaxDistance(geometry g1, geometry g2);
```

3D

返回 2 个几何体之间的最大距离。 `g1` 和 `g2` 可以是任何几何类型。

If only one geometry is provided, or `g1` and `g2` are the same geometry, returns the distance between the two vertices farthest apart in that geometry.

1.5.0 版本引入。

Enhanced: 3.7.0 - support for a single geometry input.

3D

3D 几何类型

```
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');
```

14.142135623730951

☐☐

[ST_Distance](#), [ST_LongestLine](#), [ST_DFullyWithin](#)

7.12.19 ST_3DMaxDistance

ST_3DMaxDistance — 返回两个几何体 (SRS 相同) 3 维空间中的最大距离。

Synopsis

```
float ST_3DMaxDistance(geometry g1, geometry g2);
```

☐☐

返回两个几何体 (SRS 相同) 3 维空间中的最大距离。

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

✔ This function supports Polyhedral surfaces.

2.0.0 版本引入。

注意: 2.2.0 版本, 2D 和 3D 几何体 Z 坐标 Z 为 0 时返回 0。

☐☐

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point and line compared 2D point and line) Note that 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` — 几何体的鲁棒性 (robustness) 与最小清除距离 (clearance) 的函数。

Synopsis

```
float ST_MinimumClearance(geometry g);
```

几何体

(几何体) `ST_IsValid` 与 (几何体) `ST_IsSimple` 函数返回布尔值，表示几何体是否有效 (WKT, KML, GML, GeoJSON 等) 或是否简单 (MapInfo TAB 等)。如果几何体无效或不是简单多边形，则返回 false。

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.

几何体 e 的最小清除距离，即几何体 e 的鲁棒性：

- 几何体 e 的最小清除距离 2 倍于 e 的鲁棒性。
- 几何体 e 的鲁棒性 e 的最小清除距离。

几何体 e 的最小清除距离 (即几何体 e 的鲁棒性) 由 `ST_MinimumClearance` 函数返回。

To avoid validity issues caused by precision loss, **ST_ReducePrecision** can reduce coordinate precision while ensuring that polygonal geometry remains valid.

2.3.0 版本引入。

几何体

```
SELECT ST_MinimumClearance('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))');
```

```
0.00032
```

几何体

ST_MinimumClearanceLine, **ST_Crosses**, **ST_Dimension**, **ST_Intersects**

7.12.21 ST_MinimumClearanceLine

`ST_MinimumClearanceLine` — 返回几何体 2 倍的最小清除距离，即几何体的鲁棒性。

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 3.10.0

2.3.0 支持。GEOS 3.6.0 支持。

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 类型，返回 2 维欧几里得距离。对于 ST_Geography 类型，返回 0 维欧几里得距离。对于 ST_Geography 类型，返回 ST_Length 返回的结果。对于 ST_Geography 类型，返回 ST_Length 返回的结果。

For geography types, the calculations are performed using the inverse geodesic problem, where perimeter units are in meters. The spheroid is specified by the SRID; when no SRID is provided, WGS84 is used. If use_spheroid = false, then calculations will approximate a sphere instead of a spheroid.

对于 ST_Perimeter2D 类型，返回 2 维欧几里得距离。

✓ 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

PostGIS: 2.0.0 支持。

XXXXXXXXXXXXXXXXXXXXX. XXX EPSG:2249 XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX.

122.63074400009504

845.227713366825

1. 本圖係根據 1:50,000 比例尺之地形圖繪製，其坐標系統為 WGS84 坐標系統。

```
per_meters | per_ft
-----+-----
37.37904625414049 | 122.63466618812497
```

```
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);`

Computes the 2D perimeter of the geometry `geomA`.



Note
This function is an alias for `ST_Perimeter`. `ST_Perimeter` is the preferred function name.

[ST_Perimeter](#)

7.12.24 ST_3DPerimeter

`ST_3DPerimeter` — Computes the 3D perimeter of the geometry `geomA`.

Synopsis

`float ST_3DPerimeter(geometry geomA);`

Computes the 3D perimeter of the geometry `geomA`.



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

PostGIS 2.0.0 introduced `ST_Perimeter3D`.

例

计算多边形的周长

```
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
105.46579359767418		105.43299727218763		105.43299727218763

例

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);

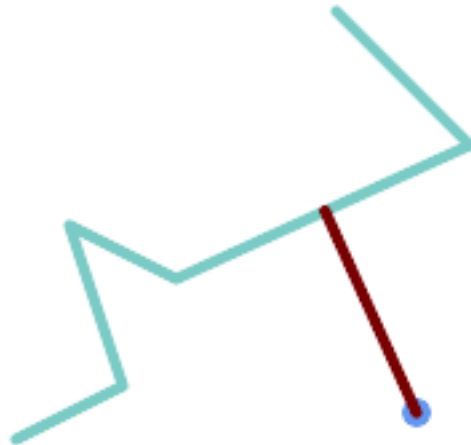
例

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 添加支持。

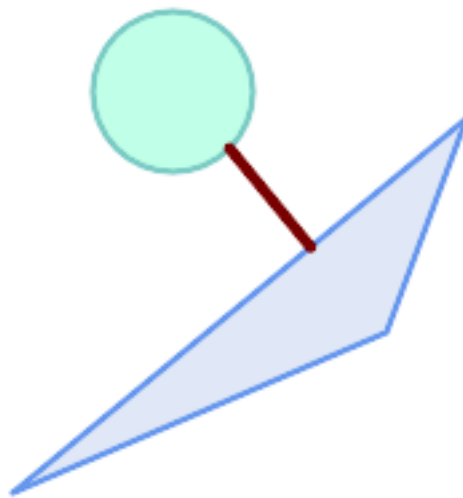
测试



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 llinewkt;
```

```
LINESTRING(131.59149149528952 101.89887534906195,101.21320343559643 138.78679656440357)
```


☒☒☒☒☒☒☒☒☒☒ -- 3D, 2D ☒☒☒☒☒☒☒☒

```
SELECT ST_AsEWKT(ST_3DShortestLine(line,pt)) AS shl3d_line_pt,
        ST_AsEWKT(ST_ShortestLine(line,pt)) As shl2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)>::geometry As pt,
        'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)>:: ←
geometry As line
      ) As foo;
```

shl3d_line_pt | shl2d_line_pt

LINESTRING(54.69937988676193 128.93502291722837 11.547586950660556,100 100 30) | ↩

LINESTRING(50 75, 50 74) 3D, 2D

```
SELECT ST_AsEWKT(ST_3DShortestLine(poly, mline)) As shl3d,
       ST_AsEWKT(ST_ShortestLine(poly, mline)) As shl2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,
100 100 5, 175 150 5))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125
100 1, 175 155 1),
       (1 10 2, 5 20 1))') As mline ) As foo;
```

shl3d | ↩
shl2d

LINESTRING(40.45454545454545 53.63636363636364 5,40.45454545454545 53.63636363636364 5) | LINESTRING(20 40,20 40)



ST 3DClosestPoint, ST 3DDistance, ST LongestLine, ST ShortestLine, ST 3DMaxDistance

7.13 Overlay Functions

7.13.1 ST_ClipByBox2D

ST ClipByBox2D — Computes the portion of a geometry falling within a rectangle.

Synopsis

```
geometry ST_ClipByBox2D(geometry geom, box2d box);
```



Clips a geometry by a 2D box in a fast and tolerant but possibly invalid way. Topologically invalid input geometries do not result in exceptions being thrown. The output geometry is not guaranteed to be valid (in particular, self-intersections for a polygon may be introduced).

GEOS ☒☒☒☒☒

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.

￼￼

This example relies on the 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 snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see [ST_ReducePrecision](#).

GEOS ￼￼￼￼

Enhanced: 3.1.0 accept a **gridSize** parameter.

Requires GEOS $\geq 3.9.0$ to use the **gridSize** parameter.



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

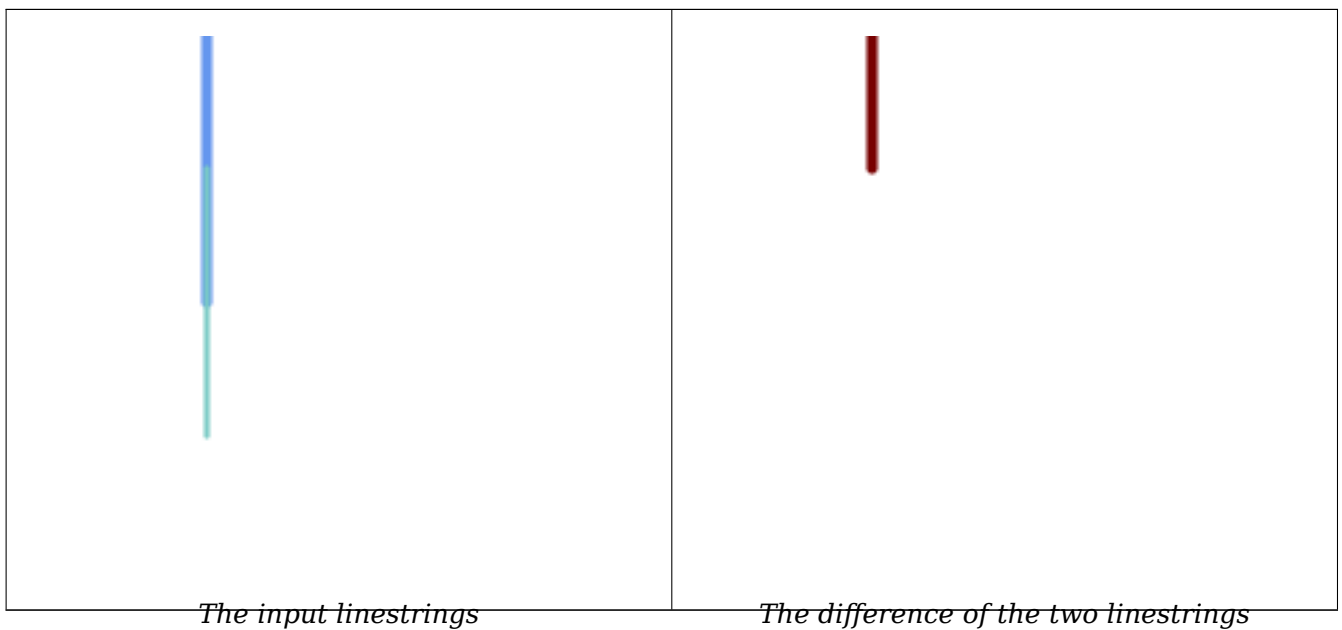


This method implements the SQL/MM specification. SQL-MM 3: 5.1.20



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

￼￼



The difference of 2D linestrings.

```
SELECT ST_AsText(ST_Difference('LINESTRING(50 100, 50 200)>::geometry,
                              'LINESTRING(50 50, 50 150)>::geometry
                              )
);
```

```
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
                              ) );
```

```
MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)
```

☒☒

ST_SymDifference, ST_Intersection, ST_Union, ST_ReducePrecision

7.13.3 ST_Intersection

ST_Intersection — Computes a geometry representing the shared portion of geometries A and B.

Synopsis

```
geometry ST_Intersection( geometry geomA , geometry geomB , float8 gridSize = -1 );
geography ST_Intersection( geography geogA , geography geogB );
```



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 snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, 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 SFGCAL based [ST_3DIntersection](#) which does a proper 3D intersection for 3D geometries. Although this function works with Z-coordinate, it does an averaging of Z-Coordinate.

GEOS 国国国国国国

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));
```

```
POINT EMPTY
```

```
SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 )':: ↵
    geometry));
```

```
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: 2.5D geometries

Note this is not a true 3D intersection. It uses 2.5D geometries (geometries with Z ordinates, but where calculations are performed in 2D). 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;
```

```
LINESTRING Z (1 1 8,0.5 0.5 8,0 0 10)
```

☒☒

[ST_3DIntersection](#), [ST_Difference](#), [ST_Union](#), [ST_ClipByBox2D](#), [ST_Dimension](#), [ST_Dump](#), [ST_Force2D](#), [ST_SymDifference](#), [ST_Intersects](#), [ST_Multi](#), [ST_ReducePrecision](#)

7.13.4 ST_MemUnion

ST_MemUnion — Aggregate function which unions geometries in a memory-efficient but slower way

Synopsis

geometry **ST_MemUnion**(geometry set geomfield);



An aggregate function that unions the input geometries, merging them to produce a result geometry with no overlaps. The output may be a single geometry, a MultiGeometry, or a Geometry Collection.



Note

Produces the same result as **ST_Union**, but uses less memory and more processor time. This aggregate function works by unioning the geometries incrementally, as opposed to the ST_Union aggregate which first accumulates an array and then unions the contents using a fast algorithm.



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.



```
SELECT id,
       ST_MemUnion(geom) as singlegeom
FROM sometable f
GROUP BY id;
```



ST_Union

7.13.5 ST_Node

ST_Node — Nodes a collection of lines.

Synopsis

geometry **ST_Node**(geometry geom);



Returns a (Multi)LineString representing the fully noded version of a collection of linestrings. The noding preserves all of the input nodes, and introduces the least possible number of new nodes. The resulting linework is dissolved (duplicate lines are removed).

This is a good way to create fully-noded linework suitable for use as input to **ST_Polygonize**.

ST_UnaryUnion can also be used to node and dissolve linework. It provides an option to specify a gridSize, which can provide simpler and more robust output. See also **ST_Union** for an aggregate variant.



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

GEOS ☒ ☒ ☒ ☒ ☒

2.0.0 □□□□□□□□□□.

Changed: 2.4.0 this function uses GEOSNode internally instead of GEOSUnaryUnion. This may cause the resulting linestrings to have a different order and direction compared to PostGIS < 2.4.



Noding a 3D LineString which self-intersects

```
SELECT ST_AsText(ST_Node('LINESTRINGZ(0 0 0, 10 10 10, 0 10 5, 10 0 3)')::geometry)
      ) As output;
```

```
MULTILINESTRING Z ((0 0 0,5 5 4.5),(5 5 4.5,10 10 10,0 10 5,5 5 4.5),(5 5 4.5,10 0 3))
```

Noding two LineStrings which share common linework. Note that the result linework is dissolved.

```
SELECT ST_AsText(ST_Node('MULTILINESTRING ((2 5, 2 1, 7 1), (6 1, 4 1, 2 3, 2 5))':: geometry)
) As output;
```

```
MULTILINESTRING((2 5,2 3),(2 3,2 1,4 1),(4 1,2 3),(4 1,6 1),(6 1,7 1))
```



ST UnaryUnion, ST AsBinary

7.13.6 ST_Split

ST Split — Returns a collection of geometries created by splitting a geometry by another geometry.

Synopsis

```
geometry ST_Split(geometry input, geometry blade);
```



The function supports splitting a `LineString` by a `(Multi)Point`, `(Multi)LineString` or `(Multi)Polygon` boundary, or a `(Multi)Polygon` by a `LineString`. When a `(Multi)Polygon` is used as the blade, its linear components (the boundary) are used for splitting the input. The result geometry is always a collection.

This function is in a sense the opposite of `ST_Union`. Applying `ST_Union` to the returned collection should theoretically yield the original geometry (although due to numerical rounding this may not be exactly the case).

**Note**

If the the input and blade do not intersect due to numerical precision issues, the input may not be split as expected. To avoid this situation it may be necessary to snap the input to the blade first, using `ST_Snap` with a small tolerance.

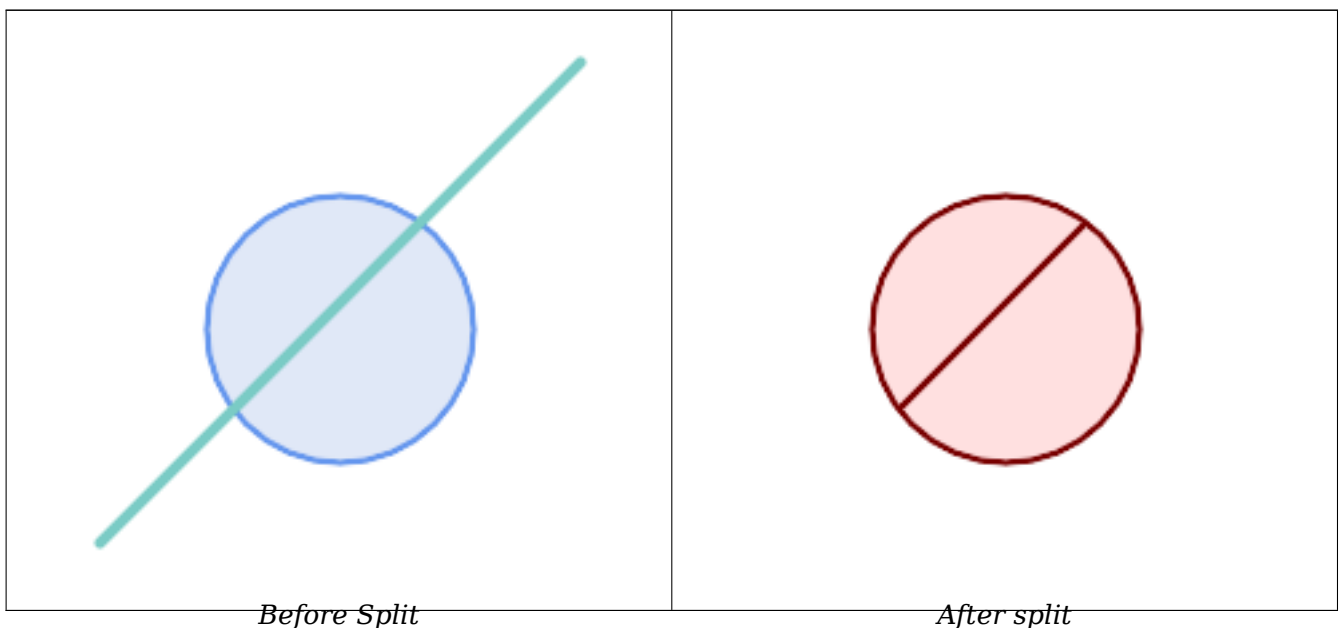
Availability: 2.0.0 requires GEOS

Enhanced: 2.2.0 support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced.

Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced.



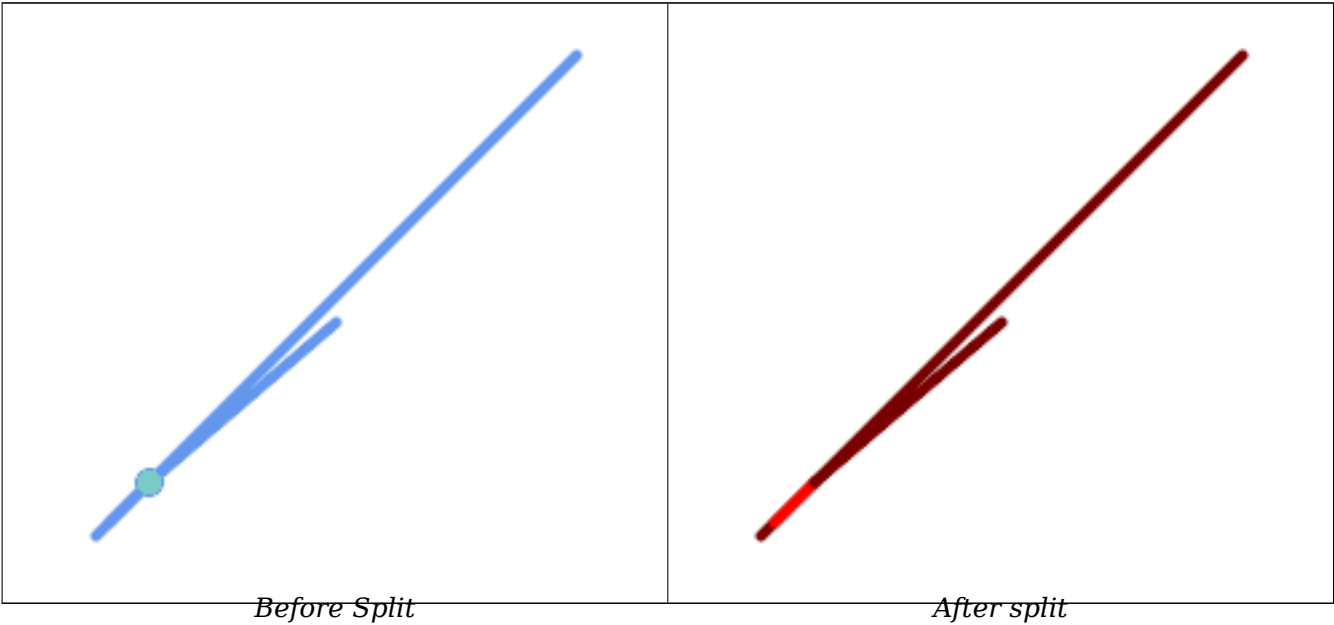
Split a Polygon by a Line.



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

```
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)));
```

```
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 snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, 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



When casting a subdivided geometry to geography, the resulting geography may differ from the original. Subdivision adds vertices in planar (geometry) space. If vertices are inserted along the boundary, they will alter the geographical representation, where edges are interpreted as geodesic segments. To minimize distortion, first densify the geography using [ST_Segmentize](#) to add geodesic vertices, then cast to geometry before subdivision.

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   .

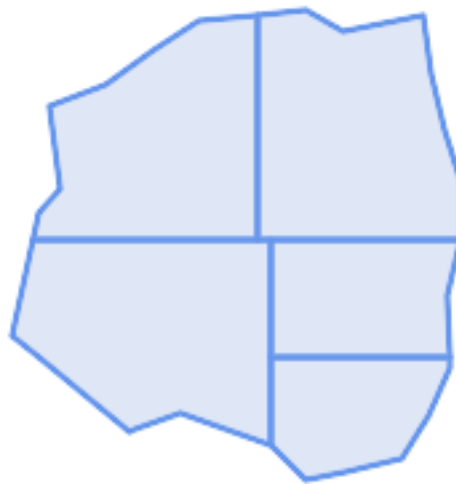
Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5.

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.

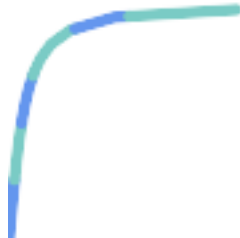


Subdivided to maximum 10 vertices

```
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((85 35,68 29,66 28,32 56,22 64,29.82608695652174 100,119 100,119 23,85 35))
2	b'' b''	POLYGON((186 52,178 34,168 18,147 13,132 10,119 23,119 56,186 56,186 52))
3	b'' b''	POLYGON((185 79,186 56,119 56,119 100,190 100,185 79))
4	b'' b''	POLYGON((40 119,36 150,57 158,75 171,92 182,114 184,114 100,29.82608695652174 100,32 110,40 119))
5	b'' b''	POLYGON((132 186,146 178,176 184,179 162,184 141,190 122,190 100,114 100,114 184,132 186))

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. Densification minimizes the impact of changes to the geography representation of a geometry when subdividing.



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 snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see [ST_ReducePrecision](#).

GEOS 

Enhanced: 3.1.0 accept a `gridSize` parameter.

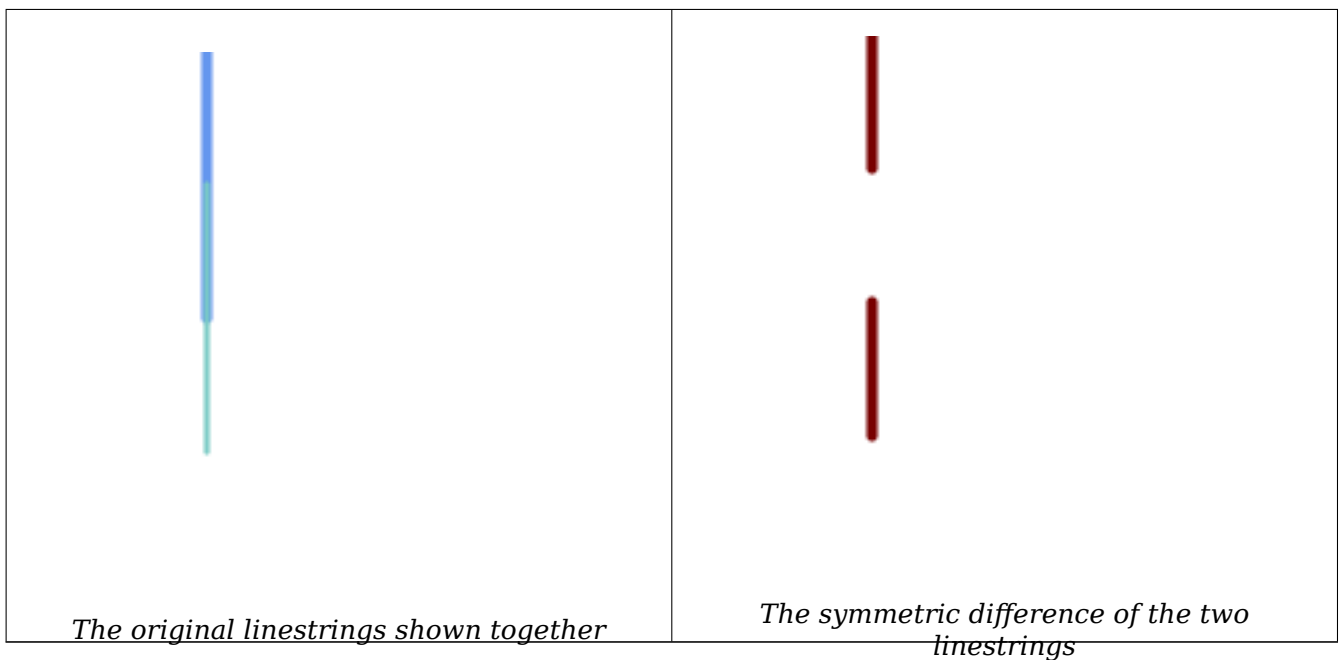
Requires GEOS $\geq 3.9.0$ to use the `gridSize` parameter

 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.





This example shows a 2D-safe symmetric difference of two LineStrings.

```
SELECT ST_AsText(ST_SymDifference(ST_GeomFromText('LINESTRING(50 100, 50 200)'),
    ST_GeomFromText('LINESTRING(50 50, 50 150)')
  )
);
```

```
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)')))
```

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

☒☒

[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);

国国

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 snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, 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 国国国国国国国国国国国国.

国国

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);
```

国国

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 snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, 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));
```

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

```
SELECT ST_AsText(ST_Union('POINT(1 2)::geometry, 'POINT(1 2)::geometry));
```

```
POINT(1 2)
```

3D example - sort of supports 3D (and with mixed geometry types!)

```
SELECT ST_AsEWKT(ST_Union(geom))
FROM (
  SELECT 'POLYGON Z((-7 4.2 5, -7.1 4.2 5, -7.1 4.3 5, -7 4.2 5))'::geometry geom
  UNION ALL
  SELECT 'POINT Z(5 5 5)'::geometry geom
  UNION ALL
  SELECT 'POINT Z(-2 3 1)'::geometry geom
  UNION ALL
  SELECT 'LINESTRING Z(5 5 5, 10 10 10)'::geometry geom
) AS foo;
```

```
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)))
```

This 3D example does not mix 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;
```

```
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)))
```

This example unions an array built from a query.

```
SELECT ST_Union(ARRAY(SELECT geom FROM sometable));
```

This example unions an array of linestrings.

```
SELECT ST_AsText(ST_Union(ARRAY[
  ST_GeomFromText('LINESTRING(1 2, 3 4)'),
  ST_GeomFromText('LINESTRING(3 4, 4 5)')
])) AS wktunion;
```

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

☒☒

[ST_CoverageUnion](#), [ST_UnaryUnion](#), [ST_MemUnion](#), [ST_Collect](#), [ST_Intersection](#), [ST_Difference](#), [ST_SymDifference](#), [ST_LineMerge](#)

7.14 几何函数

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);
```

几何

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' : defaults to 'both'. '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 when 'left' or 'right' are specified.

Note



For geography this is a thin wrapper around the geometry implementation. It determines a planar spatial reference system that best fits the bounding box of the geography object (trying UTM, Lambert Azimuthal Equal Area (LAEA) North/South pole, and finally Mercator). The buffer is computed in the planar space, and then transformed back to WGS84. This may not produce the desired behavior if the input object is much larger than a UTM zone or crosses the dateline



Note

Buffer can handle invalid inputs and the output is always a valid polygonal geometry. Buffering by distance 0 is sometimes used as a way of repairing invalid polygons. **ST_MakeValid** is more suitable for this process as it can handle multi-polygons.



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



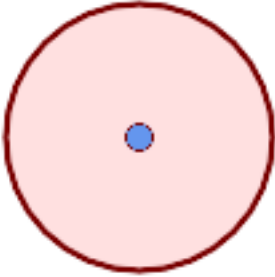
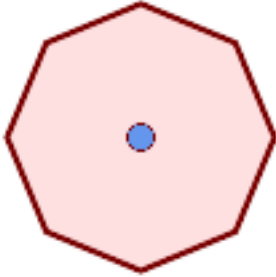
Note
This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry.

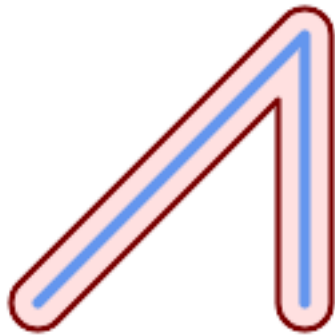
Enhanced: 2.5.0 - ST_Buffer geometry support was enhanced to allow for side buffering specification `side=both|left|right`.

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

GEOS

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1, s2.1.1.3](#)
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.30

 <i>quad_segs=8</i> () <pre>SELECT ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 'quad_segs=8');</pre>	 <i>quad_segs=2</i> () <pre>SELECT ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 'quad_segs=2');</pre>



endcap=round join=round (圆角圆角)

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



endcap=square

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



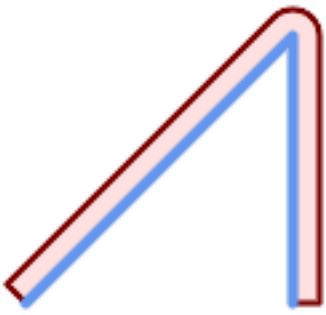
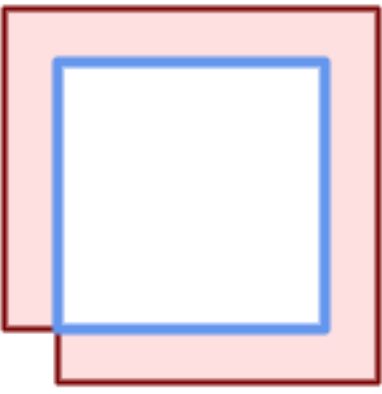
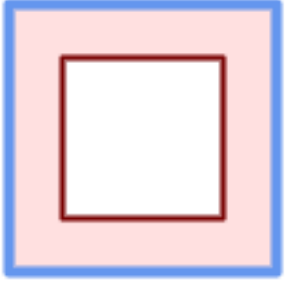
join=bevel

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



join=mitre mitre_limit=5.0 (圆角圆角圆角圆角圆角圆角)

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

 <i>side=left</i> <pre>SELECT ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)') , 10, 'side=left');</pre>	 <i>side=right</i> <pre>SELECT ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)') , 10, 'side=right');</pre>
 <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, less smooth circle using only 2 points per quarter circle is an octagon.

The following example creates a 100 meter octagon in NAD83 longitude/latitude by transforming to the Massachusetts state plane meter projection and then buffering.

```
SELECT ST_AsText(ST_Buffer(
ST_Transform(ST_SetSRID(ST_Point(-71.063526, 42.35785), 4269), 26986),
100,2)) As octagon;
```

```
POLYGON((236057.5905746494 900908.7599186979,236028.30125276805  ↵
          900838.0492405792,235957.5905746494 900808.7599186979,235886.87989653074  ↵
          900838.0492405792,235857.5905746494 900908.7599186979,235886.87989653074  ↵
          900979.4705968165,235957.5905746494 901008.7599186979,236028.30125276805  ↵
          900979.4705968165,236057.5905746494 900908.7599186979))
```

￼￼

[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](#).



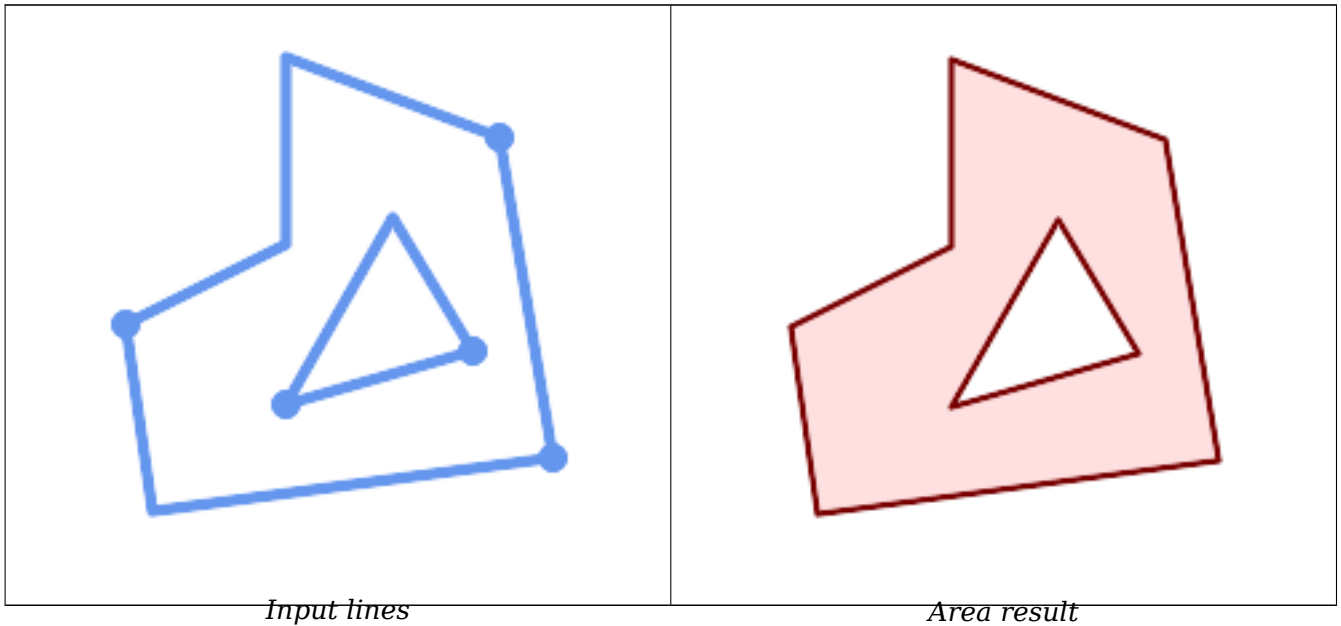
Note

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 ￼￼￼￼￼￼￼￼￼.

图



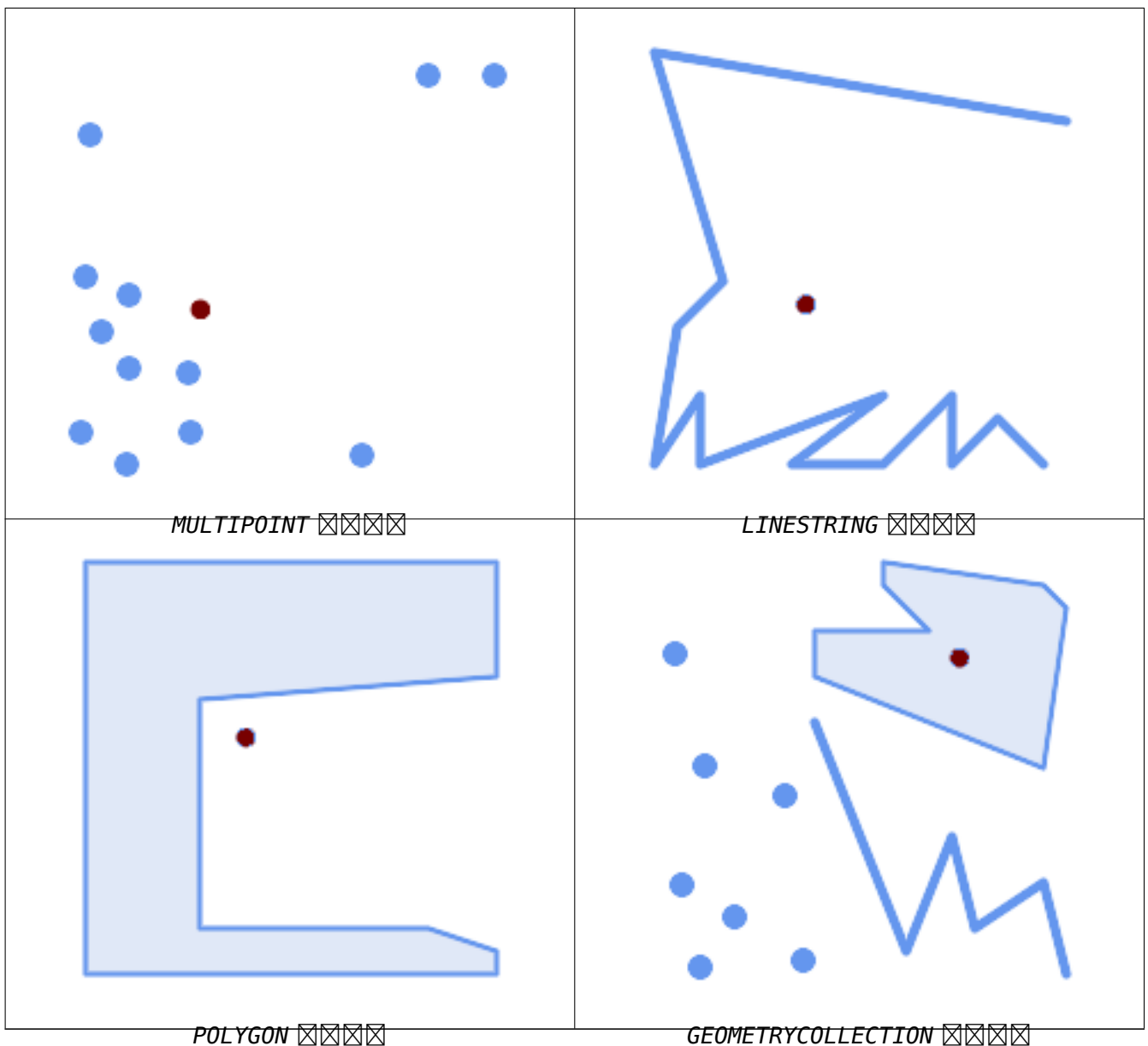
```
WITH data(geom) AS (VALUES
  ('LINESTRING (180 40, 30 20, 20 90)::geometry'),
  ('LINESTRING (180 40, 160 160)::geometry'),
  ('LINESTRING (160 160, 80 190, 80 120, 20 90)::geometry'),
  ('LINESTRING (80 60, 120 130, 150 80)::geometry'),
  ('LINESTRING (80 60, 150 80)::geometry')
)
SELECT ST_AsText(ST_BuildArea(ST_Collect(geom))) AS wkt
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
```

```
SELECT ST_AsText(ST_Centroid('MULTIPOINT ( -1 0, -1 2, -1 3, -1 4, -1 7, 0 1, 0 3, 1 1, 2 0, 6 0, 7 8, 9 8, 10 6 )'));
```

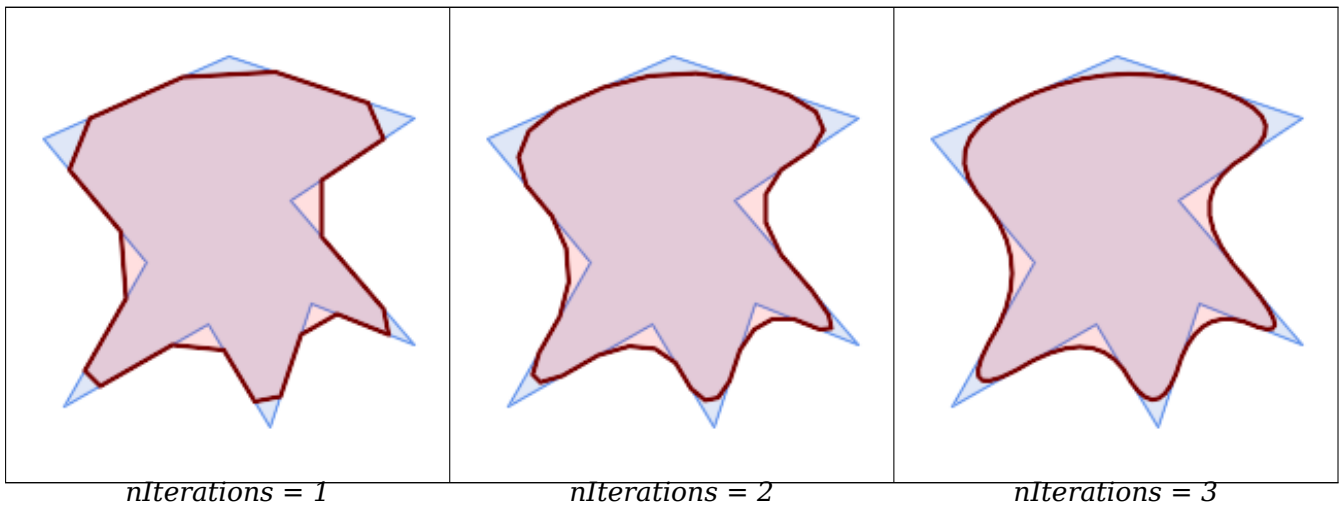
```
st_astext
```

```
-----
POINT(2.307692307692308 3.307692307692308)
(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)
```

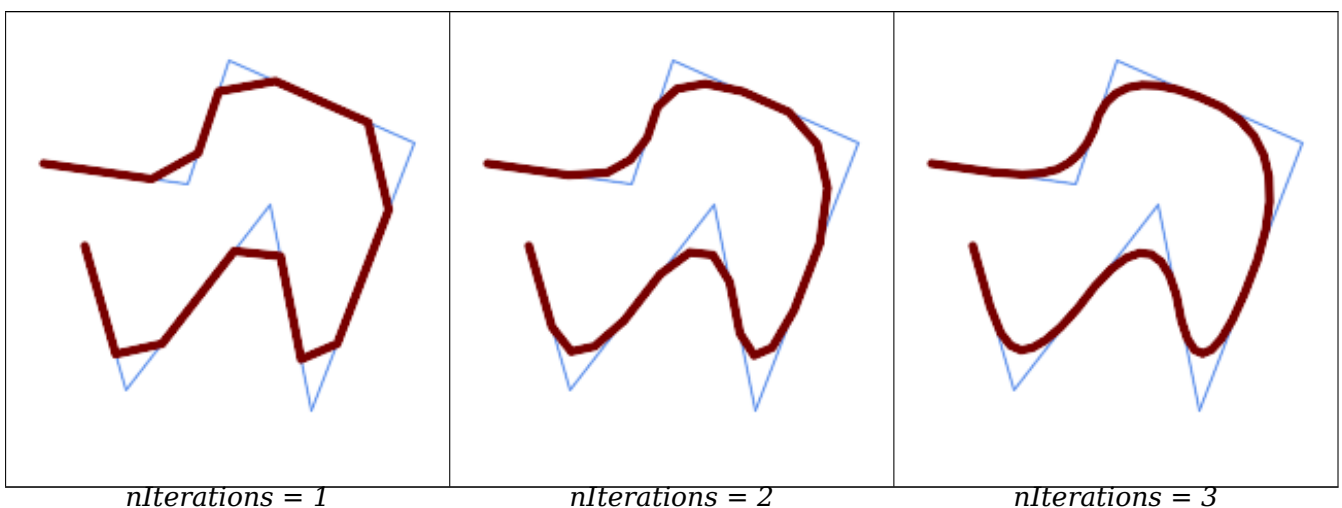
```
SELECT ST_AsText(ST_Centroid(g))
FROM ST_GeomFromText('COMPOUNDCURVE(CIRCULARSTRING(0 2, -1 1,0 0),(0 0, 0.5 0, 1 0), CIRCULARSTRING( 1 0, 2 1, 1 2),(1 2, 0.5 2, 0 2))' ) AS g;
```


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_CatmullRomSmoothing

ST_CatmullRomSmoothing — Returns a smoothed version of a geometry, using the Catmull-Rom spline algorithm

Synopsis

geometry **ST_CatmullRomSmoothing**(geometry geom, integer nSegments = 5);



Smooths a linear or polygonal geometry using the **Catmull-Rom spline** algorithm. Unlike **ST_ChaikinSmooth** this is an *interpolating* spline: the output curve passes through every original vertex. Between each pair of consecutive original vertices, nSegments - 1 new intermediate points are inserted.

The nSegments parameter controls the density of the output. With nSegments = 5 (the default), each original span is divided into 5 sub-segments (inserting 4 new points per span). The minimum value is 2.

At least 4 input vertices are required to apply smoothing; geometries with fewer vertices are returned unchanged. Points and multipoints are always returned unchanged.



Note

The output vertex count grows as $1 + (N-1) * nSegments$ for open lines, and $1 + N * nSegments$ for closed rings, where N is the number of input vertices. For large geometries, use a simplification function on the result to reduce the number of points (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.

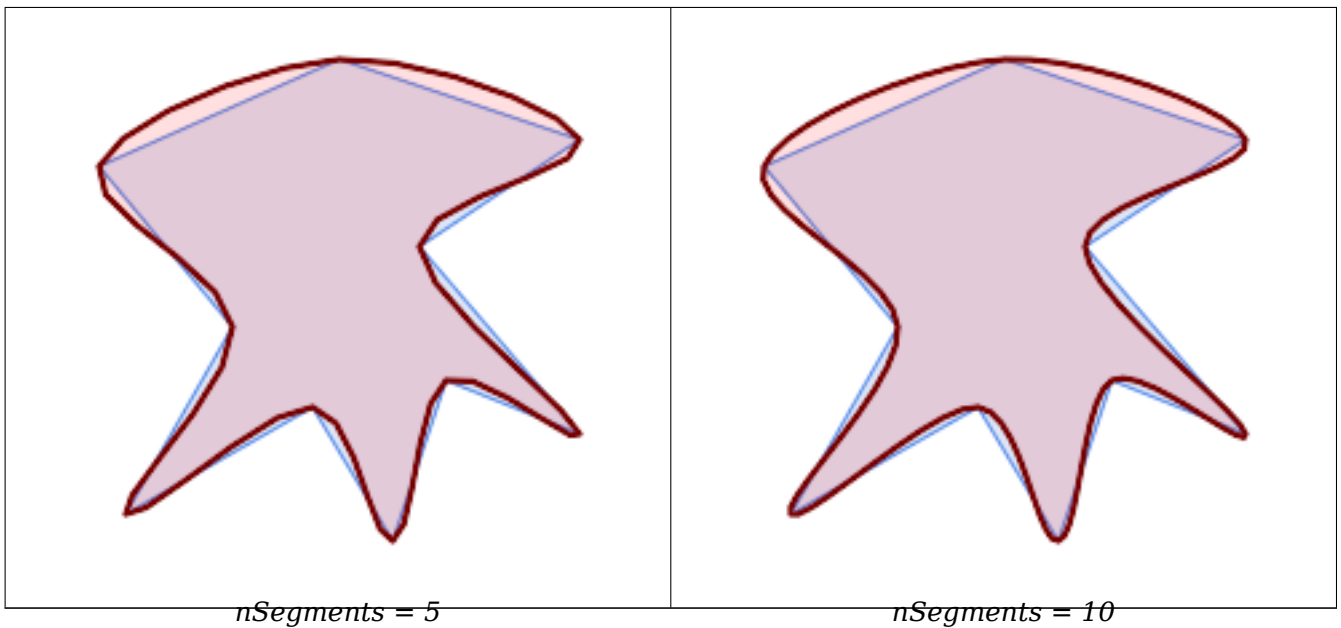
Availability: 3.6.0



Smoothing a 4-point collinear line with default nSegments=5:

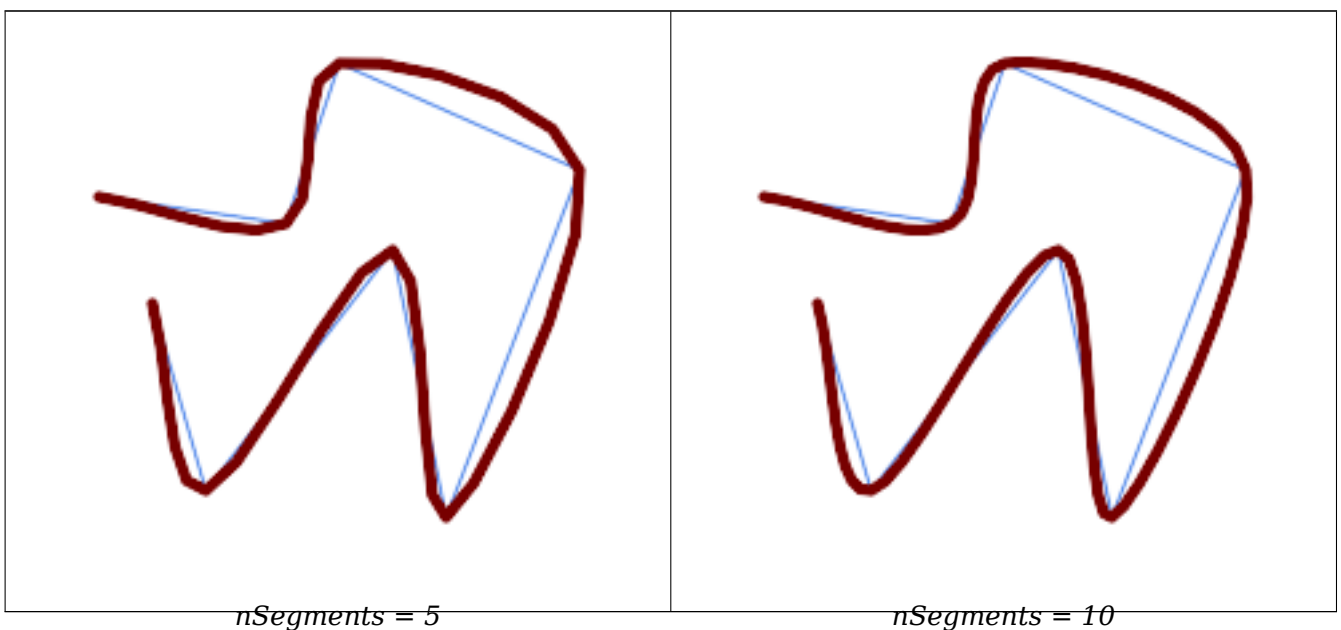
```
SELECT ST_AsText(ST_CatmullRomSmoothing('LINESTRING(0 0, 5 0, 10 0, 15 0)'));  
  
LINESTRING(0 0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 0,10 0,11 0,12 0,13 0,14 0,15 0)
```

Smoothing a Polygon using nSegments = 5 and 10:



```
SELECT ST_CatmullRomSmoothing('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(5, 10, 5) );
```

Smoothing a LineString using nSegments = 5 and 10:



```
SELECT ST_CatmullRomSmoothing('LINESTRING (10 140, 80 130, 100 190, 190 150, 140 20, 120 120, 50 30, 30 100)',
generate_series(5, 10, 5) );
```

☒☒

[ST_ChaikinSmoothing](#), [ST_Simplify](#), [ST_SimplifyPreserveTopology](#), [ST_SimplifyVW](#)

7.14.6 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);
```

国国

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.

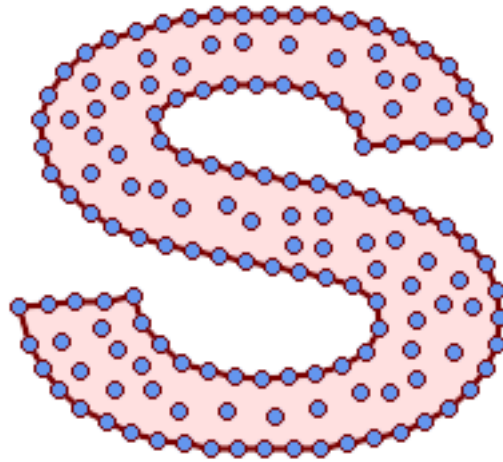
For point and linear inputs, the hull will enclose all the points of the inputs. For polygonal inputs, the hull will enclose all the points of the input *and also* all the areas covered by the input. If you want a point-wise hull of a polygonal input, convert it to points first using **ST_Points**.

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+

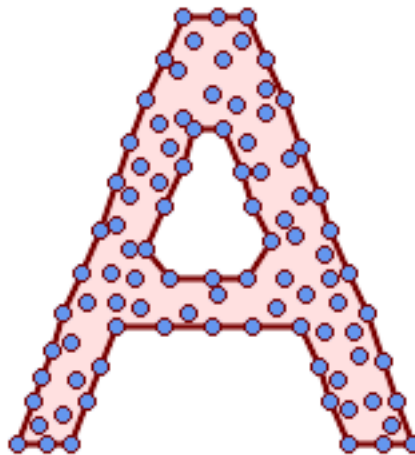
☒☒

*Concave Hull of a MultiPoint*

```

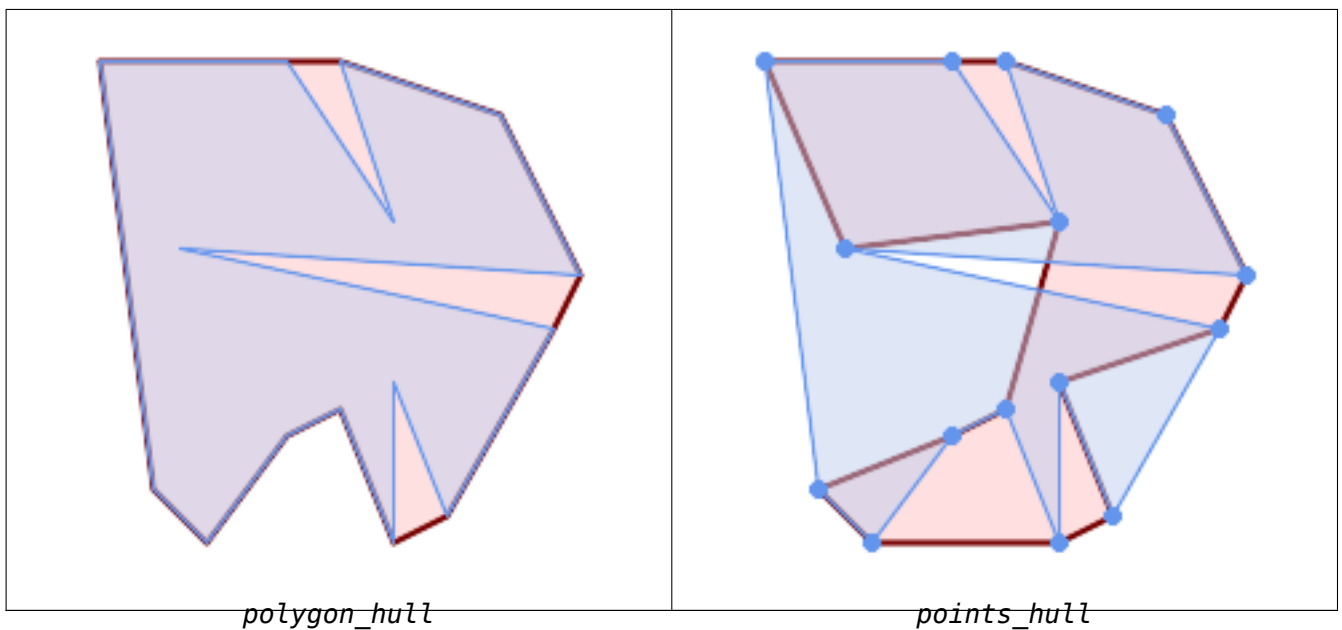
SELECT ST_AsText(ST_ConcaveHull('MULTIPOINT ((10 72), (53 76), (56 66), (63 58), (71 51), (81 48), (91 46), (101 45), (111 46), (121 47), (131 50), (140 55), (145 64), (144 74), (135 80), (125 83), (115 85), (105 87), (95 89), (85 91), (75 93), (65 95), (55 98), (45 102), (37 107), (29 114), (22 122), (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.

This example estimates an infected area from 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](#)

7.14.7 ST_ConvexHull

`ST_ConvexHull` — Computes the convex hull of a geometry.

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

ST_ConvexHull

Computes the convex hull of a geometry. The convex hull is the smallest convex geometry that encloses all geometries in the input.

One can think of the convex hull as the geometry obtained by wrapping an rubber band around a set of geometries. This is different from a **concave hull** which is analogous to "shrink-wrapping" the geometries. A convex hull is often used to determine an affected area based on a set of point observations.

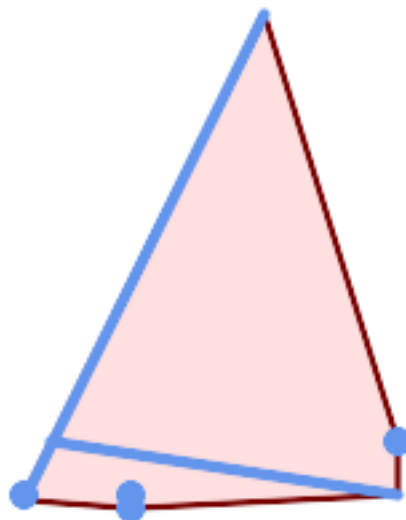
In the general case the convex hull is a Polygon. The convex hull of two or more collinear points is a two-point LineString. The convex hull of one or more identical points is a Point.

This is not an aggregate function. To compute the convex hull of a set of geometries, use **ST_Collect** to aggregate them into a geometry collection (e.g. `ST_ConvexHull(ST_Collect(geom))`).

GEOS 3.10.0

- ✓ This method implements the **OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3**
- ✓ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.16
- ✓ This function supports 3d and will not drop the z-index.

Diagram



Convex Hull of a MultiLineString and a MultiPoint

```
SELECT ST_AsText(ST_ConvexHull(ST_Collect(ST_GeomFromText('MULTILINESTRING((100 190,10 8) ↵
    ,(150 10, 20 30))'),
    ST_GeomFromText('MULTIPOINT(50 5, 150 30, 50 10, 10 10)')
    )));
---st_astext--
POLYGON((50 5,10 8,10 10,100 190,150 30,150 10,50 5))
```

Using with `ST_Collect` to compute the convex hulls of geometry sets.

This example returns an estimated infected area from 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](#)

7.14.8 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 ☐☐☐☐☐☐☐☐☐☐☐☐☐.

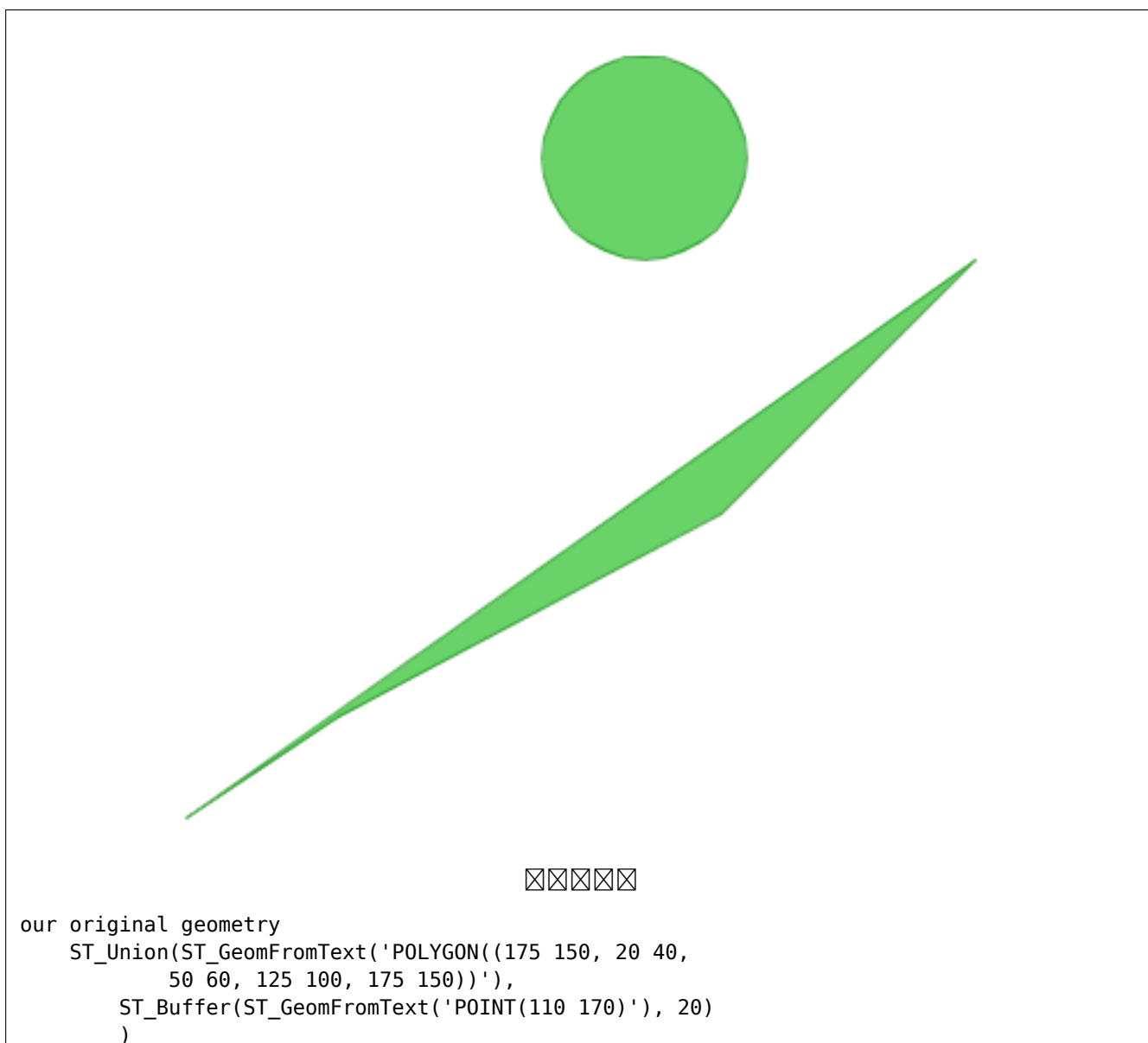


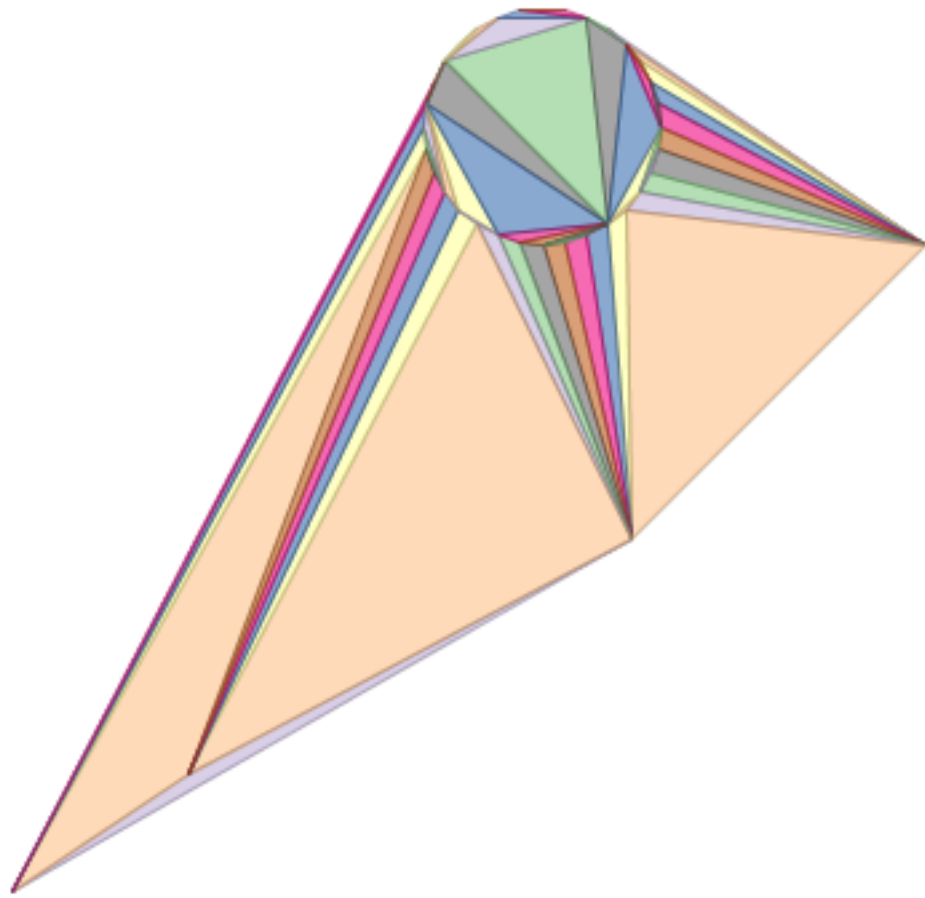
This function supports 3d and will not drop the z-index.



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

☐☐

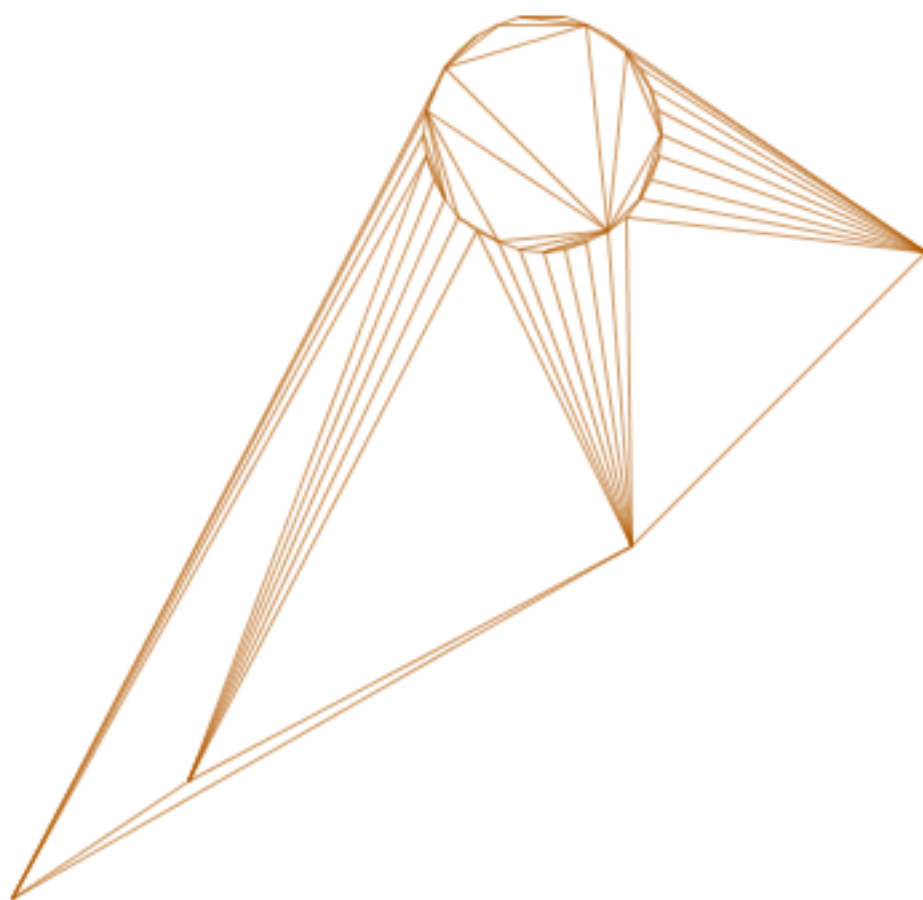




繁體中文 *ST_DelaunayTriangles*: 繁體中文

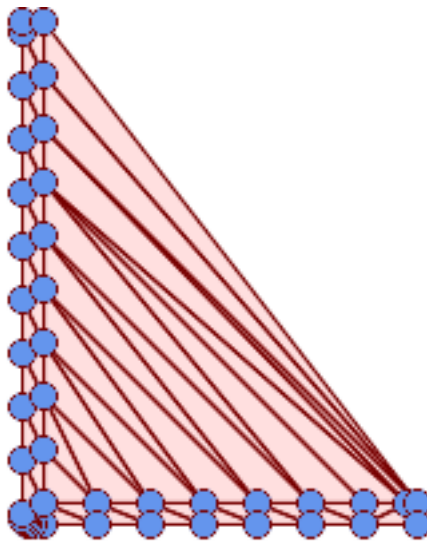
geometries overlaid multilinestring triangles

```
SELECT
  ST_DelaunayTriangles(ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
    50 60, 125 100, 175 150))'),
    ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
  ))
As dtriag;
```



Delaunay triangles as a MultiLineString

```
SELECT
  ST_DelaunayTriangles(ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
    50 60, 125 100, 175 150))'),
    ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
  ),0.001,1)
As  dtriag;
```



Delaunay triangles of 45 points as 55 triangle polygons

this produces a table of 42 points that form an L shape

```
SELECT (ST_DumpPoints(ST_GeomFromText('MULTIPOINT(14 14,34 14,54 14,74 14,94 14,114 14,134 14,150 14,154 14,154 6,134 6,114 6,94 6,74 6,54 6,34 6,14 6,10 6,8 6,7 7,6 8,6 10,6 30,6 50,6 70,6 90,6 110,6 130,6 150,6 170,6 190,6 194,14 194,14 174,14 154,14 134,14 114,14 94,14 74,14 54,14 34,14 14)'))).geom
      INTO TABLE l_shape;
```

output as individual polygon triangles

```
SELECT ST_AsText((ST_Dump(geom)).geom) As wkt
FROM ( SELECT ST_DelaunayTriangles(ST_Collect(geom)) As geom
FROM l_shape) As foo;
```

wkt

```
POLYGON((6 194,6 190,14 194,6 194))
POLYGON((14 194,6 190,14 174,14 194))
POLYGON((14 194,14 174,154 14,14 194))
POLYGON((154 14,14 174,14 154,154 14))
POLYGON((154 14,14 154,150 14,154 14))
POLYGON((154 14,150 14,154 6,154 14))
```

Example using vertices with Z values.

3D multipoint

```
SELECT ST_AsText(ST_DelaunayTriangles(ST_GeomFromText('MULTIPOINT Z(14 14 10, 150 14 100,34 6 25, 20 10 150)')))) As wkt;
```

wkt

```
GEOMETRYCOLLECTION Z (POLYGON Z ((14 14 10,20 10 150,34 6 25,14 14 10)),
  POLYGON Z ((14 14 10,34 6 25,150 14 100,14 14 10)))
```

☒☒

[ST_VoronoiPolygons](#), [ST_TriangulatePolygon](#), [ST_ConstrainedDelaunayTriangles](#), [ST_VoronoiLines](#), [ST_Con](#)

7.14.9 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;
```

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

图例

[ST_SetEffectiveArea](#), [ST_SimplifyVW](#)

7.14.10 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);

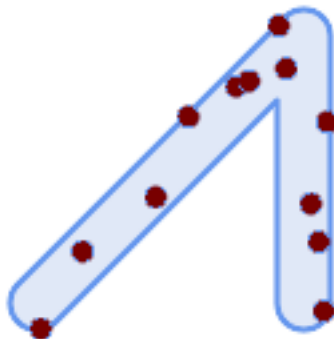
图例

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;
```


☒☒

ST_DumpPoints

7.14.11 ST_GeometricMedian

ST_GeometricMedian — 返回 MultiPoint 几何体的 (median) 几何体。

Synopsis

geometry **ST_GeometricMedian** (geometry geom, float8 tolerance = NULL, int max_iter = 10000, boolean fail_if_not_converged = false);

☒☒

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.

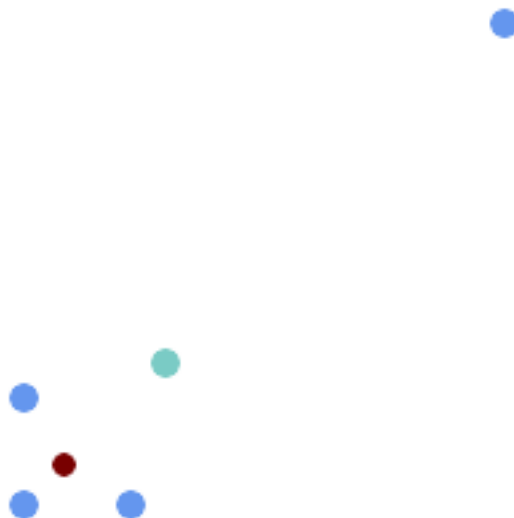
2.3.0 添加了对 M 的支持。

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.

☒☒



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

7.14.12 ST_LineMerge

ST_LineMerge — Return the lines formed by sewing together a MultiLineString.

Synopsis

```
geometry ST_LineMerge(geometry amultilinestring);
geometry ST_LineMerge(geometry amultilinestring, boolean directed);
```

￼

Returns a LineString or MultiLineString formed by joining together the line elements of a MultiLineString. Lines are joined at their endpoints at 2-way intersections. Lines are not joined across intersections of 3-way or greater degree.

If **directed** is TRUE, then ST_LineMerge will not change point order within LineStrings, so lines with opposite directions will not be merged



Note

Only use with MultiLineString/LineStrings. Other geometry types return an empty GeometryCollection

GEOS ￼￼￼￼

Enhanced: 3.3.0 accept a directed parameter.

Requires GEOS >= 3.11.0 to use the directed parameter.

1.1.0 ￼￼￼￼￼￼￼.

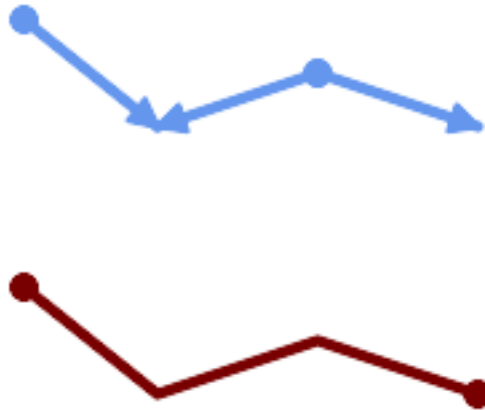


Warning

This function strips the M dimension.

图

Cardinality 2

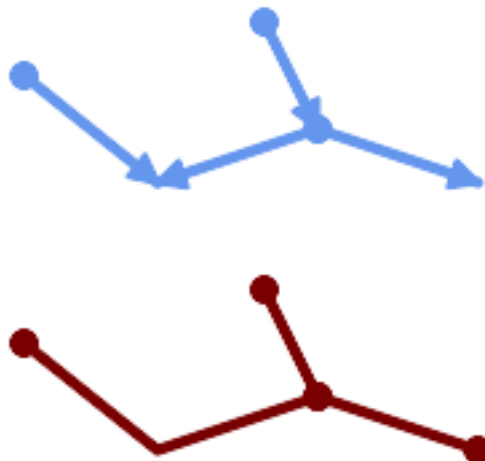


Nodes with cardinality 2 are merged away.

```
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)
```

Cardinality 3 or More



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))
```


Synopsis

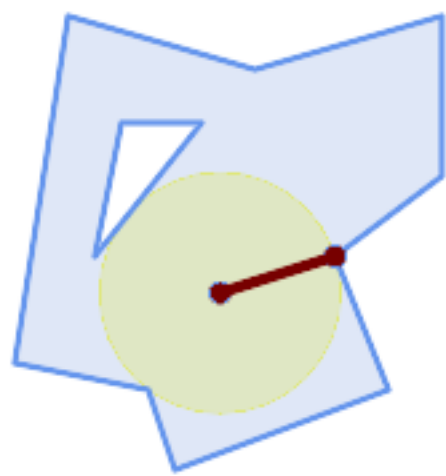
(geometry, geometry, double precision) **ST_MaximumInscribedCircle**(geometry geom);

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

- center - center point of the circle
- nearest - a point on the geometry nearest to the center
- radius - radius of the circle

For polygonal inputs, the circle is inscribed within the boundary rings, using the internal rings as boundaries. For linear and point inputs, the circle is inscribed within the convex hull of the input, using the input lines and points as further boundaries.

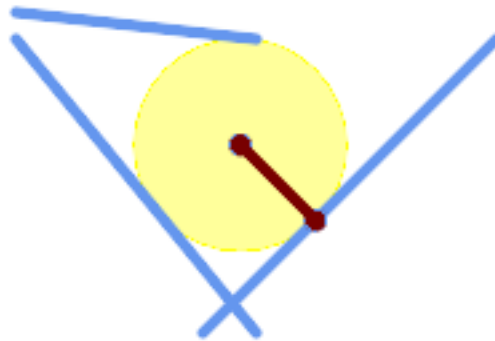
Availability: 3.1.0.
Requires GEOS >= 3.9.0.



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

```
SELECT radius, ST_AsText(center) AS center, ST_AsText(nearest) AS nearest
FROM ST_MaximumInscribedCircle('POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90,
160 40, 80 10, 70 40, 20 50, 40 180),
(60 140, 50 90, 90 140, 60 140))');
```

radius	center	nearest
45.18896951053177	POINT(96.9287109375 76.3232421875)	POINT(61.64205411585366 104.55256764481707)



Maximum inscribed circle of a multi-linestring. Center, nearest point, and radius are returned.

☒☒

[ST_MinimumBoundingRadius](#), [ST_LargestEmptyCircle](#)

7.14.14 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

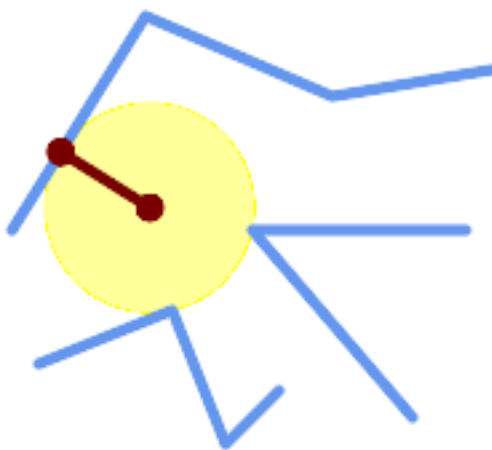
To find the largest empty circle in the interior of a polygon, see [ST_MaximumInscribedCircle](#).

Availability: 3.4.0.

Requires GEOS >= 3.9.0.

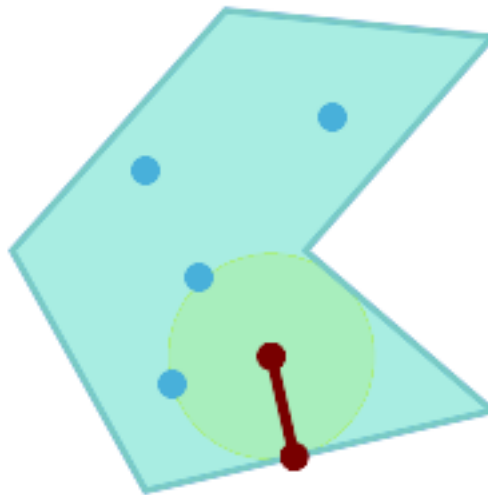
图例

```
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) ←
  0,
  'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'::geometry
);
```



Largest Empty Circle within a set of points, constrained to lie in a polygon. The constraint polygon boundary must be included as an obstacle, as well as specified as the constraint for the circle center.

ST MinimumBoundingRadius

7.14.15 ST_MinimumBoundingCircle

ST MinimumBoundingCircle — Returns the smallest circle polygon that contains a geometry.

Synopsis

```
geometry ST_MinimumBoundingCircle(geometry geomA, integer num segs per qt circ=48);
```



Returns the smallest circle polygon that contains a geometry.

Note



Note!

Note: The minimum bounding circle of the ST MinimumBoundingRadius geometry is used to determine if the geometry intersects 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**.

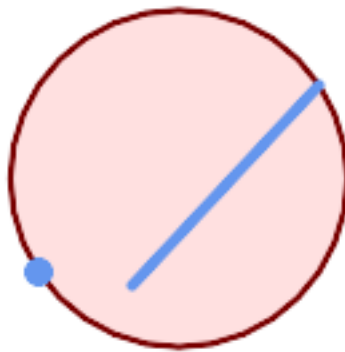
$\square$ (Roeck) $\square\square\square\square\square\square\square\square\square.$

GEOS ☐☐☐☐☐

1.4.0 ☒☒☒☒☒☒☒☒☒☒☒.

图例

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



图例显示了疾病类型及其最小包围圆。图中显示了 8 个疾病类型。

```
SELECT ST_AsText(ST_MinimumBoundingCircle(ST_Collect(ST_GeomFromText('LINESTRING(55 75,125 150)'),
                                ST_Point(20, 80))), 8
        ) As wktmbc;
```

```
POLYGON((72.5 178.4024474346694,84.86920388664029 177.18418718546332,96.76306620464646 173.57622349601382,107.72451249529237 167.71720838753708,117.33230052487836 159.83230052487838,125.21720838753708 150.22451249529237,131.07622349601382 139.26306620464646,134.68418718546332 127.3692038866403,135.9024474346694 115,134.68418718546332 102.63079611335971,131.07622349601382 90.73693379535354,125.21720838753708 79.77548750470766,117.33230052487838 70.16769947512164,107.72451249529237 62.282791612462916,96.76306620464648 56.42377650398619,84.86920388664032 52.815812814536685,72.50000000000001 51.59755256533059,60.13079611335969 52.815812814536685,48.236933795353536 56.42377650398618,37.27548750470765 62.28279161246291,27.667699475121637 70.16769947512162,19.782791612462923 79.77548750470763,13.923776503986204 90.73693379535355,10.315812814536692 102.63079611335968,9.097552565330588 114.99999999999999,10.315812814536685 127.3692038866403,13.923776503986197 139.2630662046465,19.78279161246291 150.22451249529234,27.667699475121623 159.83230052487835,37.275487504707634 167.71720838753708,48.236933795353494 173.5762234960138,60.13079611335967 177.18418718546332,72.49999999999999 178.4024474346694))
```

图例

ST_Collect, ST_MinimumBoundingRadius, ST_LargestEmptyCircle, ST_LongestLine

7.14.16 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(26284.84180271327 65267.114509082545)	247.43604559140655

返回

ST_Collect, **ST_MinimumBoundingCircle**, **ST_LongestLine**

7.14.17 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.

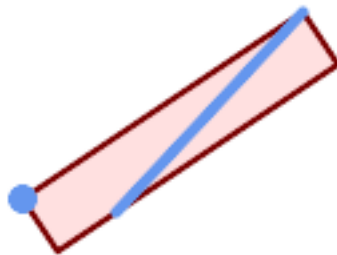
Availability: 2.5.0.

Requires GEOS >= 3.6.0.

☒☒

```
SELECT ST_AsText(ST_OrientedEnvelope('MULTIPOINT ((0 0), (-1 -1), (3 2))'));
```

```
POLYGON((3 2,-1 -1,-1.12 -0.84,2.88 2.16,3 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;
```

```
POLYGON((20 80,125 150,138.0769230769241 130.38461538461573,33.07692307692291 ↵
60.384615384615195,20 80))
```

☒☒

ST_Envelope **ST_MinimumBoundingCircle**

7.14.18 ST_OffsetCurve

ST_OffsetCurve — Returns an offset line at a given distance and side from an input line.

Synopsis

geometry **ST_OffsetCurve**(geometry line, float signed_distance, text style_parameters=“);

☒☒

Return an offset line at a given distance and side from an input line. All points of the returned geometries are not further than the given distance from the input geometry. Useful for computing parallel lines about a center line.

For positive distance the offset is on the left side of the input line and retains the same direction. For a negative distance it is on the right side and in the opposite direction.

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

- 'quad_segs=#' : 圆弧 (quarter circle) 的段数 (默认 8)
- 'join=round|mitre|bevel' : 端点连接方式 ('round' (round), 'mitre' (mitre), 'bevel' (bevel)). 'mitre_limit' 控制 'mitre' 的延伸长度。
- 'mitre_limit=#.#' : 控制 'mitre' 的延伸长度 (默认 5)。'mitre_limit' 控制 'miter_limit' 的延伸长度。

GEOS 3.11

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 简体中文版。

Enhanced: 2.5 - added support for GEOMETRYCOLLECTION and MULTILINESTRING



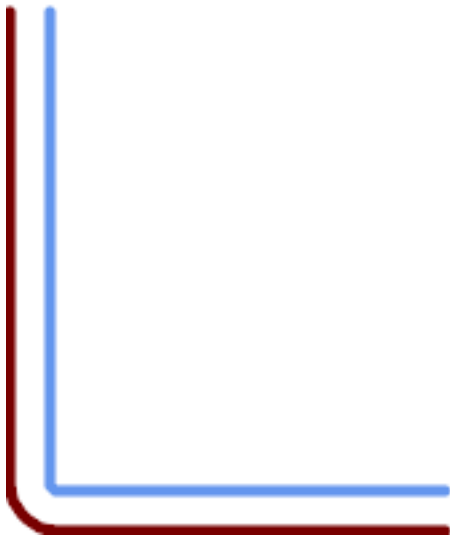
Note

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

SQL

ST_OffsetCurve(geometry, distance, [options]).

```
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)
```

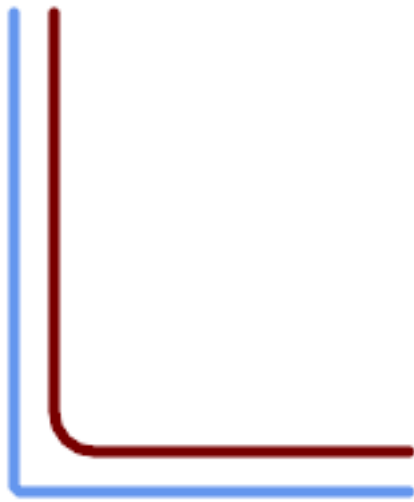


-15, 'quad_segs=4
join=round' -15

```
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)
```

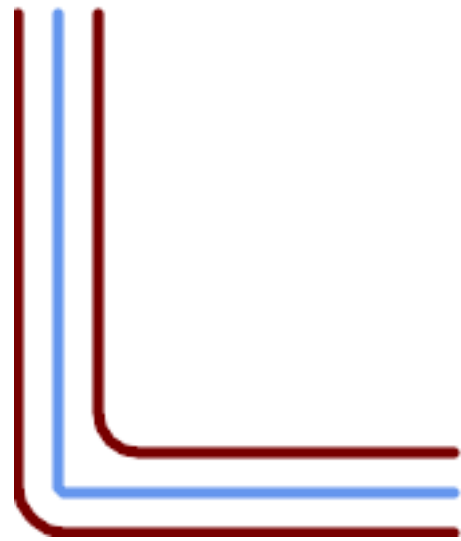


偏移距离为 15 单位，向外
偏移。 $-30 + 15 = -15$
单位。

```
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)
```



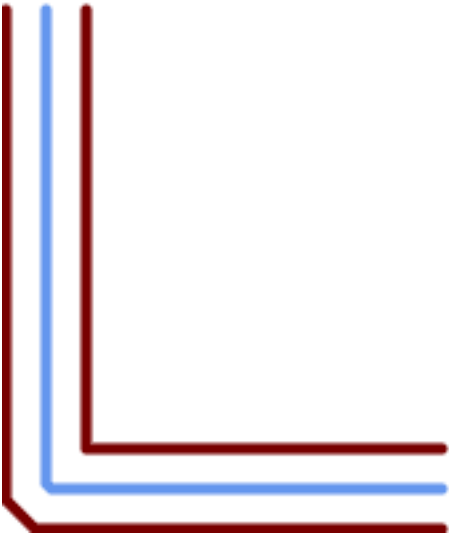
偏移距离为 15 单位，向外
偏移。 $-30 + 15 = -15$
单位。

```
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

7.14.19 ST_PointOnSurface

ST_PointOnSurface — Computes a point guaranteed to lie in a polygon, or on a geometry.

Synopsis

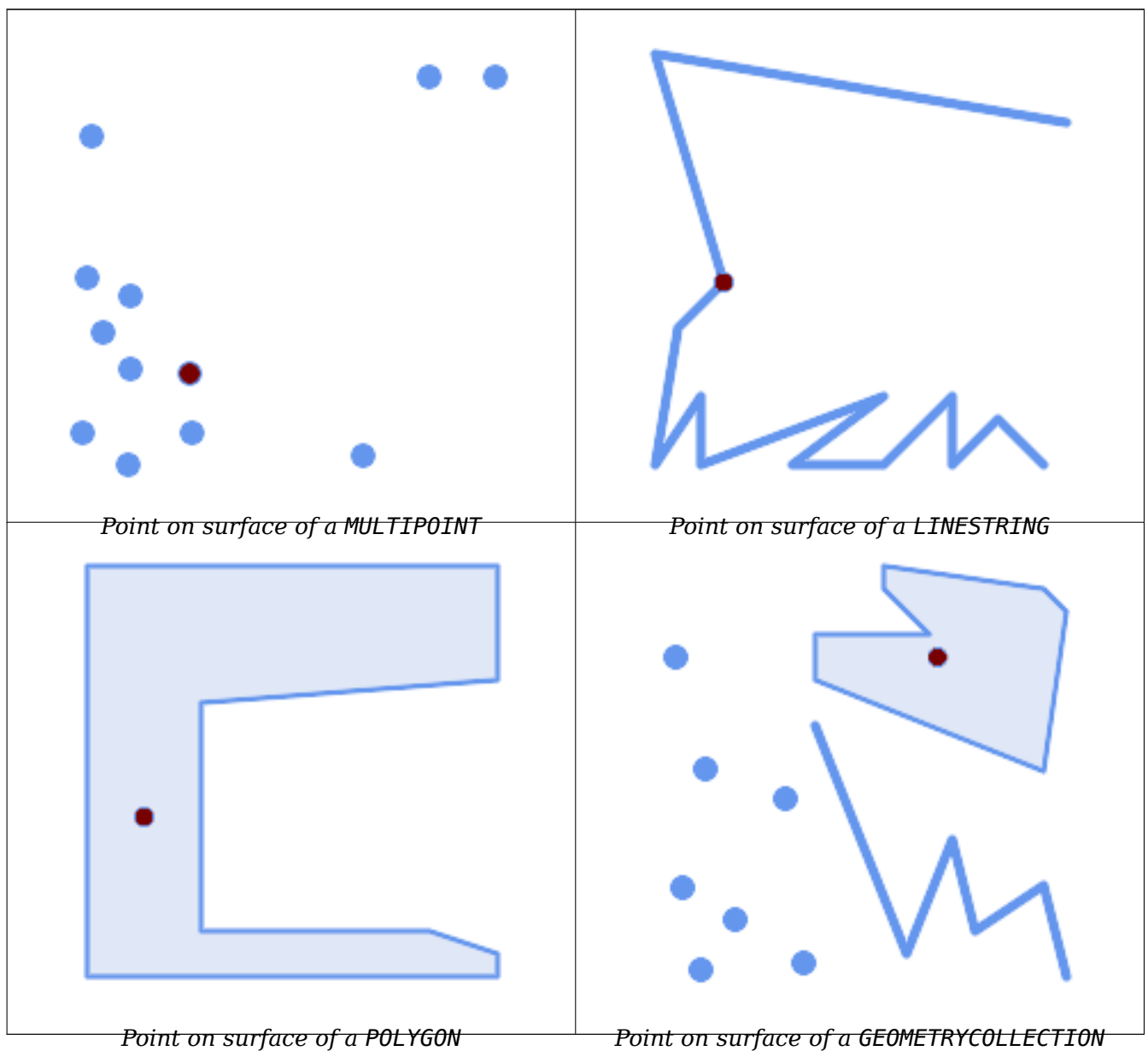
geometry **ST_PointOnSurface**(geometry g1);

☒

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)
```

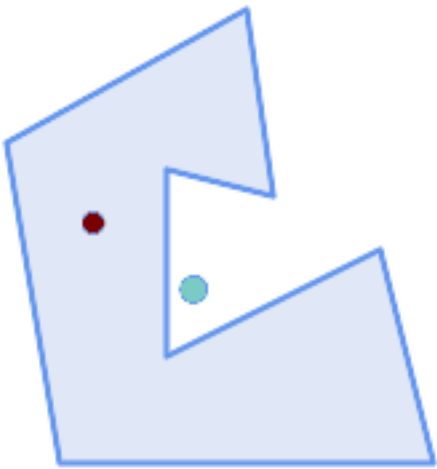


```
SELECT ST_AsText(ST_PointOnSurface('LINESTRING(0 5, 0 10)::geometry));
-----
POINT(0 5)

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**

7.14.20 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);
```

注意

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.



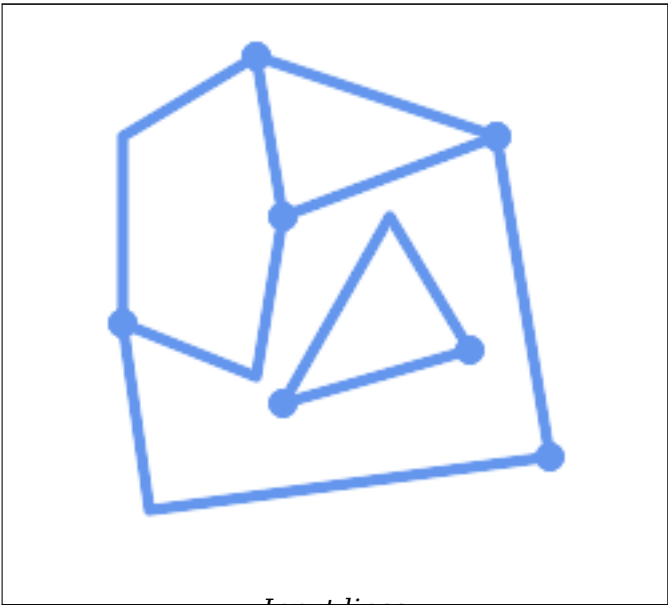
Note

GeometryCollections can be difficult to handle with external tools. Use [ST_Dump](#) to convert the polygonized result into separate polygons.

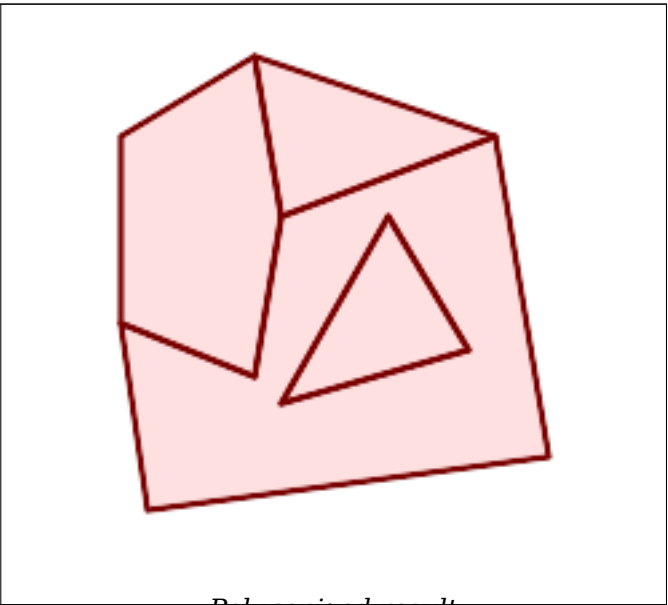
GEOS 简体中文

1.0.0RC1 简体中文.

注意



Input lines



Polygonized result

```
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((80 60,120 130,150 80,80 60)),POLYGON((80 130,70 70,20 90,20 160,70 190,80 130)),POLYGON((160 160,80 130,70 190,160 160)))
```

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**

7.14.21 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.

☐☐

```
SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 0.1));
```

```
POINT(1.4 19.3)
```

```
SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 1.0));
```

```
POINT(1 19)
```

```
SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 10));
```

```
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));
```

```
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));
```

```
MULTIPOLYGON(((40 40,50 10,10 10,40 40)),((60 60,70 30,40 40,60 60)))
```

☐☐

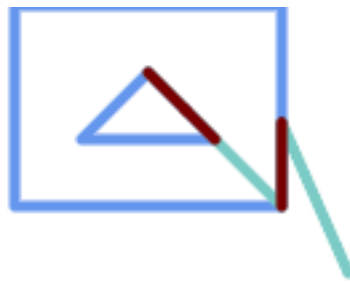
ST_SnapToGrid, **ST_Simplify**, **ST_SimplifyVW**

7.14.22 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: 共享路径 (Shared Paths)

```
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)')
)
)

GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125),
(101 150,90 161),(90 161,76 175)),MULTILINESTRING EMPTY)
```

same example but linestring orientation flipped

```
SELECT ST_AsText(ST_SharedPaths(ST_GeomFromText('LINESTRING(76 175,90 161,126 125,126 156.25,151 100)'),
ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
(51 150,101 150,76 175,51 150))')
)
)

GEOMETRYCOLLECTION(MULTILINESTRING EMPTY,
MULTILINESTRING((76 175,90 161),(90 161,101 150),(126 125,126 156.25)))
```

图例

[ST_Dump](#), [ST_GeometryN](#), [ST_NumGeometries](#)

7.14.23 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);
```

简介

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

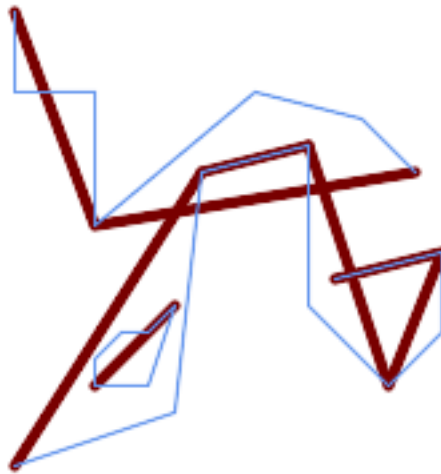
简介

A circle simplified with a medium tolerance becomes an octagon, and with a large tolerance disappears.

```
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_gone,
       (ST_Simplify(geom, 100) is null) AS np100_geometrygoesaway
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) As geom) AS t;
```

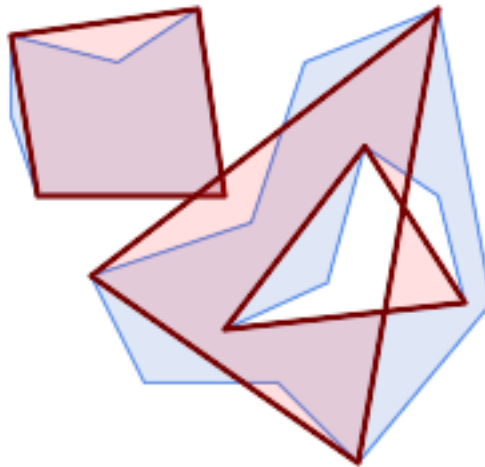
np_before		np01_notbadcircle		np05_notquitecircle		np1_octagon		np10_gone		↔
np100_geometrygoesaway										
-----+-----+-----+-----+-----+-----+-----										
49		33		17		9		null		t

Simplifying a set of lines. Lines may intersect after simplification.



```
SELECT ST_Simplify('MULTILINESTRING ((20 180, 20 150, 50 150, 50 100, 110 150, 150 140, 170 120), (20 10, 80 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))',
```

Simplifying a MultiPolygon. Polygonal results may be invalid.



```
SELECT ST_Simplify('MULTIPOLYGON (((90 110, 80 180, 50 160, 10 170, 10 140, 20 110, 90 110) ), ((40 80, 100 100, 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)))',
```

☒☒

[ST_IsSimple](#), [ST_SimplifyPreserveTopology](#), [ST_SimplifyVW](#), [ST_CoverageSimplify](#), [Topology](#) [ST_Simplify](#)

7.14.24 ST_SimplifyPreserveTopology

`ST_SimplifyPreserveTopology` — Returns a simplified and valid representation of a geometry, using the Douglas-Peucker algorithm.

Synopsis

geometry **ST_SimplifyPreserveTopology**(geometry geom, float tolerance);

Computes a simplified representation of a geometry using a variant of the **Douglas-Peucker algorithm** which limits simplification to ensure the result has the same topology as the input. The simplification tolerance is a distance value, in the units of the input SRS. Simplification removes vertices which are within the tolerance distance of the simplified linework, as long as topology is preserved. The result will be valid and simple if the input is.

The function can be called with any kind of geometry (including GeometryCollections), but only line and polygon elements are simplified. For polygonal inputs, the result will have the same number of rings (shells and holes), and the rings will not cross. Ring endpoints may be simplified. For linear inputs, the result will have the same number of lines, and lines will not intersect if they did not do so in the original geometry. Endpoints of linear geometry are preserved.



Note This function does not preserve boundaries shared between polygons. Use **ST_CoverageSimplify** if this is required.

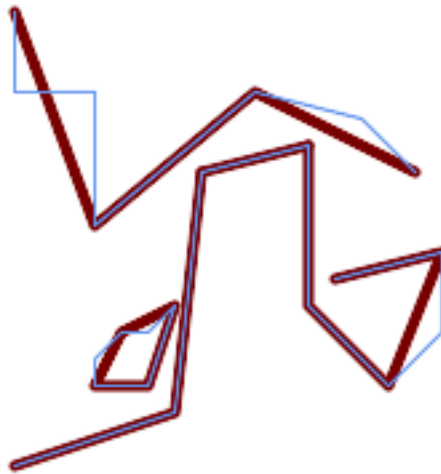
GEOS
1.3.3

For the same example as **ST_Simplify**, **ST_SimplifyPreserveTopology** prevents the geometry from disappearing.

```
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_minimal,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 100)) AS np100_minimal
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) AS geom) AS t;
```

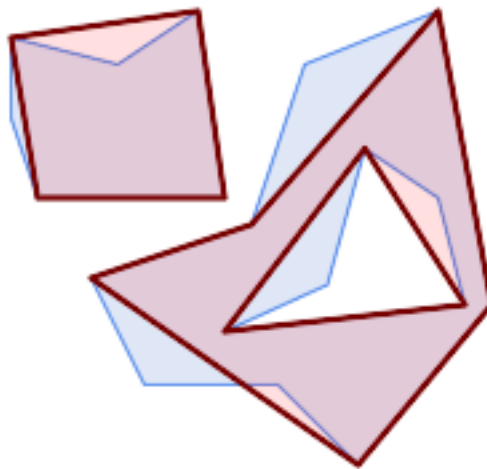
np_before	np01_notbadcircle	np05_notquitecircle	np1_octagon	np10_minimal	
49	33	17	9	4	

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 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](#)

7.14.25 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);

几何

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.

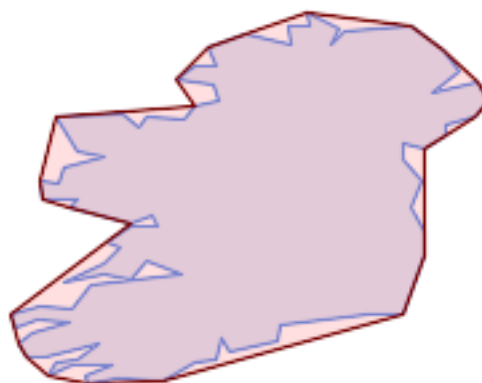
To get better results with geometries that contain relatively long line segments, it might be necessary to "segmentize" the input, as shown below.

GEOS 几何

Availability: 3.3.0.

Requires GEOS >= 3.11.0.

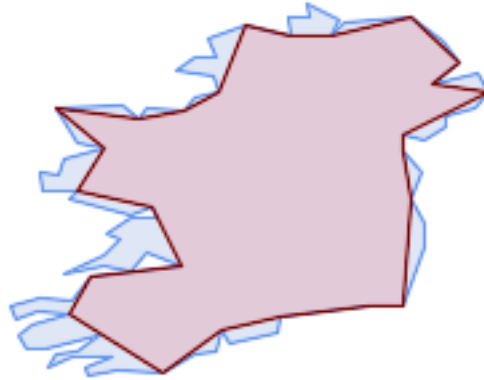
几何



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

7.14.26 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



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

Note!

[illegible]

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.

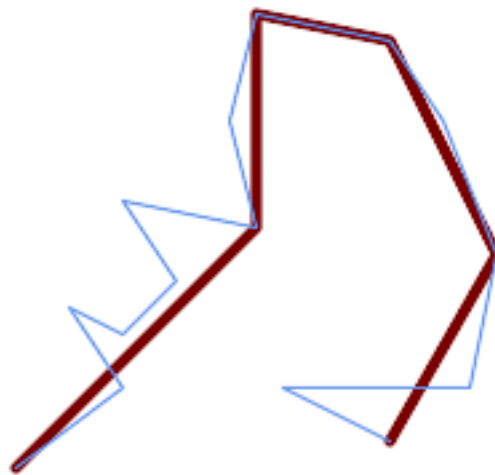
图例

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;
```

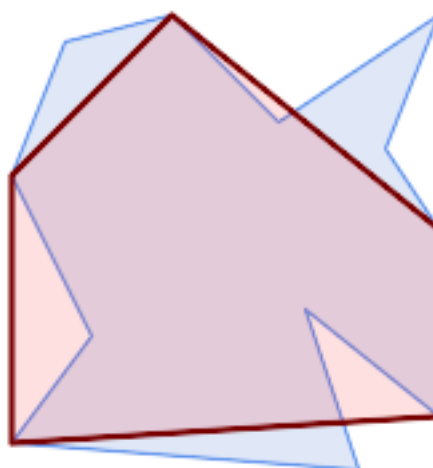
```
LINESTRING(5 2,7 25,10 10)
```

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);
```

图例

[ST_SetEffectiveArea](#), [ST_Simplify](#), [ST_SimplifyPreserveTopology](#), [ST_CoverageSimplify](#), [Topology](#) [ST_Simpl](#)

7.14.27 ST_SetEffectiveArea

`ST_SetEffectiveArea` — Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.

Synopsis

geometry **ST_SetEffectiveArea**(geometry geom, float threshold = 0, integer set_area = 1);

图例

图例展示了使用 `ST_SetEffectiveArea` 函数对几何体进行简化后的结果。图中显示了原始几何体（M 型）和简化后的几何体。简化后的几何体在保持拓扑结构的同时，减少了不必要的顶点，从而提高了效率。

图例展示了使用 `ST_SetEffectiveArea` 函数对几何体进行简化后的结果。图中显示了原始几何体（M 型）和简化后的几何体。简化后的几何体在保持拓扑结构的同时，减少了不必要的顶点，从而提高了效率。

图例展示了使用 `ST_SetEffectiveArea` 函数对几何体进行简化后的结果。图中显示了原始几何体（M 型）和简化后的几何体。简化后的几何体在保持拓扑结构的同时，减少了不必要的顶点，从而提高了效率。



Note

图例展示了使用 `ST_SetEffectiveArea` 函数对几何体进行简化后的结果。图中显示了原始几何体（M 型）和简化后的几何体。简化后的几何体在保持拓扑结构的同时，减少了不必要的顶点，从而提高了效率。



Note

图例展示了使用 `ST_SetEffectiveArea` 函数对几何体进行简化后的结果。图中显示了原始几何体（M 型）和简化后的几何体。简化后的几何体在保持拓扑结构的同时，减少了不必要的顶点，从而提高了效率。



Note

图例展示了使用 `ST_SetEffectiveArea` 函数对几何体进行简化后的结果。图中显示了原始几何体（M 型）和简化后的几何体。简化后的几何体在保持拓扑结构的同时，减少了不必要的顶点，从而提高了效率。



Note

图例展示了使用 `ST_SetEffectiveArea` 函数对几何体进行简化后的结果。图中显示了原始几何体（M 型）和简化后的几何体。简化后的几何体在保持拓扑结构的同时，减少了不必要的顶点，从而提高了效率。

2.2.0 图例展示了使用 `ST_SetEffectiveArea` 函数对几何体进行简化后的结果。图中显示了原始几何体（M 型）和简化后的几何体。简化后的几何体在保持拓扑结构的同时，减少了不必要的顶点，从而提高了效率。

图例

图例显示了有效面积的计算结果。图例 0 显示了有效面积，图例 30 显示了有效面积。

```
select ST_AsText(ST_SetEffectiveArea(geom)) all_pts, ST_AsText(ST_SetEffectiveArea(geom,30) ←
) thrshld_30
FROM (SELECT 'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry geom) As foo;
```

all_pts	thrshld_30
LINESTRING M (5 2 3.402823466385289e+38,3 8 29,6 20 1.5,7 25 49.5,10 10 3.402823466385289e ← +38) LINESTRING M (5 2 3.402823466385289e+38,7 25 49.5,10 10 3.402823466385289e+38)	

(1 row)

图例

ST_SimplifyVW

7.14.28 ST_TriangulatePolygon

ST_TriangulatePolygon — Computes the constrained Delaunay triangulation of polygons

Synopsis

geometry **ST_TriangulatePolygon**(geometry geom);

图例

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.

图例

Triangulation of a square.

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

```

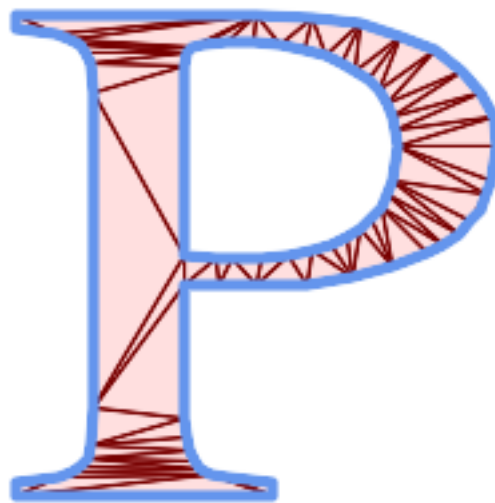
```
GEOMETRYCOLLECTION(POLYGON((0 0,0 1,1 1,0 0)),POLYGON((1 1,1 0,0 0,1 1)))

```


图

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)))'
));
```

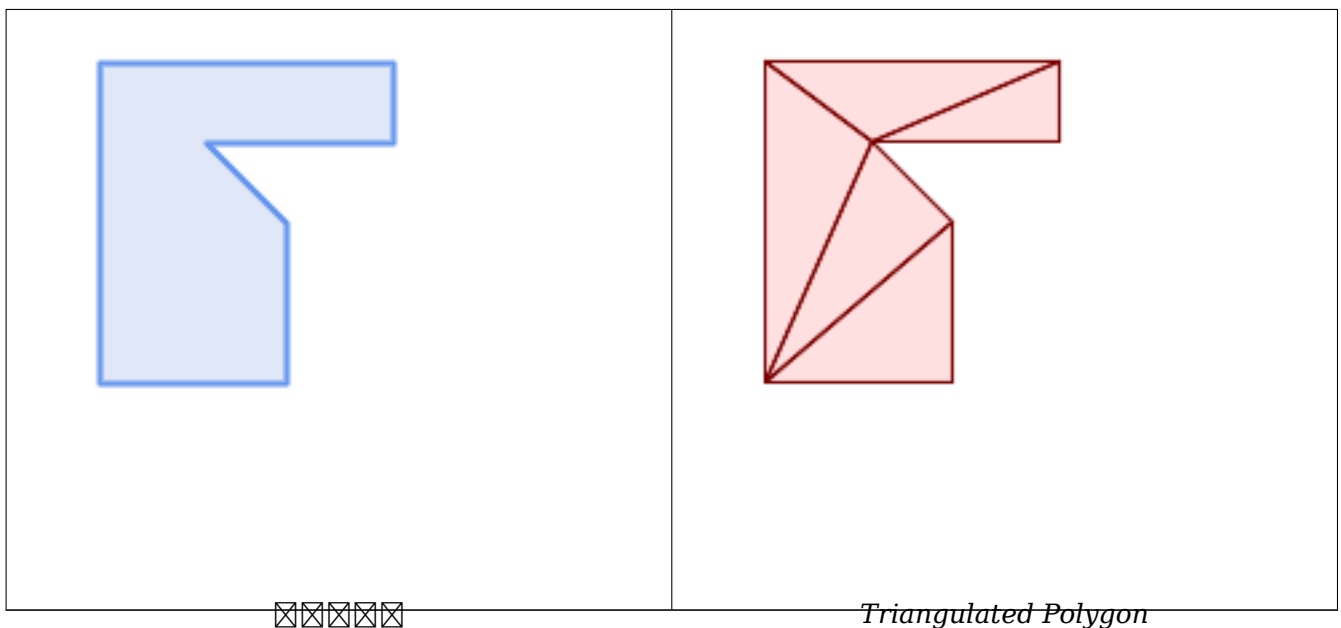


Polygon Triangulation

Same example as ST_Tessellate

```
SELECT ST_AsText(ST_TriangulatePolygon('POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160, ↵
  120 160, 120 190, 10 190 ))')::geometry
));
```

```
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](#)

7.14.29 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.

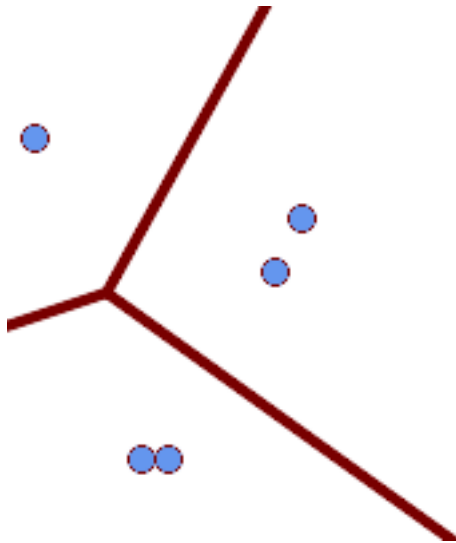
￼￼￼￼￼￼￼:

- **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 ￼￼￼￼￼￼￼.

图例



Voronoi diagram lines, with tolerance of 30 units

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

```
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

7.14.30 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.

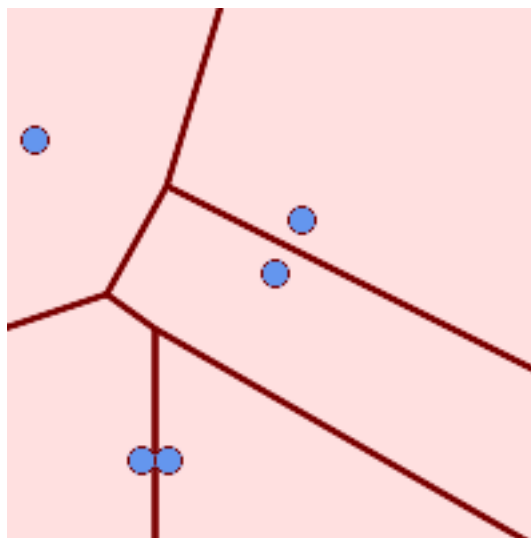
图例:

- **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 简体中文

2.3.0 简体中文.

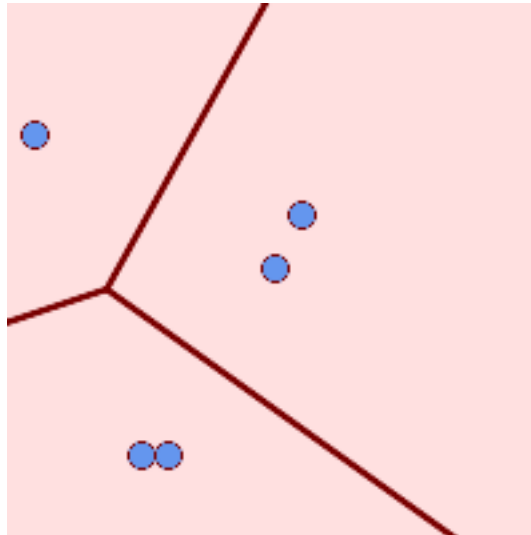
图



Points overlaid on top of Voronoi diagram

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

```
GEOMETRYCOLLECTION(POLYGON((-110 270,100.5 270,59.34782608695652 132.82608695652175,36.81818181818182 92.27272727272727,-110 43.33333333333332,-110 270)),
POLYGON((-110 -90,-110 43.33333333333332,36.81818181818182 92.27272727272727,55 79.28571428571429,55 -90,-110 -90)),
POLYGON((230 -20.71428571428572,55 79.28571428571429,36.81818181818182 92.27272727272727,59.34782608695652 132.82608695652175,230 47.50000000000001,230 -20.71428571428572)),POLYGON((230 -90,55 -90,55 79.28571428571429,230 -20.71428571428572,230 -90)),
POLYGON((230 270,230 47.50000000000001,59.34782608695652 132.82608695652175,100.5 270,230 270)))
```



Voronoi diagram, with tolerance of 30 units

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

```
GEOMETRYCOLLECTION(POLYGON((-110 270,135.55555555555557 270,36.81818181818182
  92.27272727272727,-110 43.33333333333332,-110 270)),
POLYGON((230 270,230 -45.71428571428574,36.81818181818182
  92.27272727272727,135.55555555555557 270,230 270)),
POLYGON((230 -90,-110 -90,-110 43.33333333333332,36.81818181818182 92.27272727272727,230
  -45.71428571428574,230 -90)))
```

￼￼

[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.

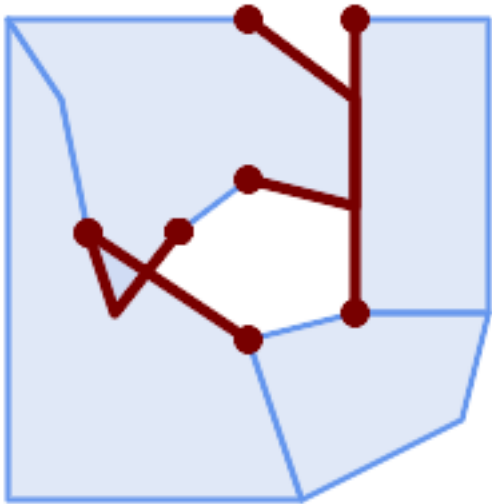
The conditions allow a valid coverage to contain holes (gaps between polygons), as long as the surrounding polygons are edge-matched. However, very narrow gaps are often undesirable. If the *tolerance* parameter is specified with a non-zero distance, edges forming narrower gaps will also be returned 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

乱乱



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

This example tests the entire table for coverage validity.

```
SELECT true = ALL (
  SELECT ST_CoverageInvalidEdges(geom) OVER () IS NULL
  FROM coverage
);
```

￼

[ST_IsValid](#), [ST_CoverageUnion](#), [ST_CoverageClean](#), [ST_CoverageSimplify](#)

7.15.2 ST_CoverageClean

ST_CoverageClean — Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.

Synopsis

geometry **ST_CoverageClean**(geometry winset geom, float8 gapMaximumWidth = 0, float8 snappingDistance = -1, text overlapMergeStrategy = 'MERGE_LONGEST_BORDER');

￼

A window function which adjusts the edges of a set of valid polygonal geometries to produce a clean coverage. Cleaning involves:

- snapping vertices and edges to remove small discrepancies and ensure common edges are identically noded
- merging overlaps into a parent polygon
- merging narrow gaps into adjacent polygons

gapMaximumWidth controls which gaps between polygons are merged. Gaps with width $\leq$ this distance are merged into an adjacent polygon.

snappingDistance controls snapping of vertices and edges. The default (-1) automatically determines a snapping distance based on the input extent. Set to 0.0 to turn off snapping.

overlapMergeStrategy specifies how overlaps are merged into a parent polygon:

- **MERGE_LONGEST_BORDER** - merges into polygon with longest common border
- **MERGE_MAX_AREA** - merges into polygon with maximum area
- **MERGE_MIN_AREA** - merges into polygon with minimum area
- **MERGE_MIN_INDEX** - merges into polygon with smallest input index (defined by order of input polygons)

The result is a clean polygonal coverage that will pass validation by [ST_CoverageInvalidEdges](#) and can be input to coverage processing functions.

Note

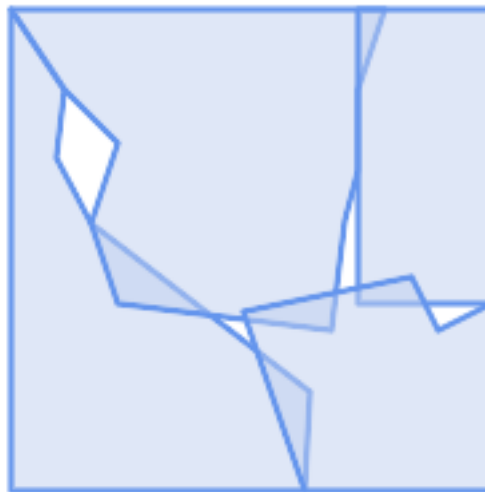
To aid in determining a maximum gap width, gaps can be computed by cleaning with `gapMaximumWidth = > 0` and using `ST_CoverageUnion` to union the result coverage. Holes in the union correspond to gaps in the original dataset. Gap widths can be measured by extracting the holes as polygons and running `ST_MaximumInscribedCircle` on them; the gap width is twice the computed radius.

Availability: 3.6.0 - requires GEOS >= 3.14.0



This example populates the input table.

```
CREATE TABLE example AS SELECT * FROM (VALUES
  (1, 'POLYGON ((10 190, 30 160, 27 134.5, 40 110, 122 47, 120 10, 10 10, 10 190))':: geometry),
  (2, 'POLYGON ((150 190, 10 190, 30 160, 50 140, 40 110, 50 80, 130 70, 135 111, 140 130, 140 160, 150 190))'::geometry),
  (3, 'POLYGON ((140 190, 190 190, 190 80, 140 80, 140 190))'::geometry),
  (4, 'POLYGON ((190 10, 120 10, 97 77, 160 90, 170 70, 190 80, 190 10))'::geometry)
) AS v(id, geom);
```



Polygons with overlaps and gaps

This example shows that the coverage is invalid.

```
SELECT ST_AsText(ST_CoverageInvalidEdges(geom) OVER ())
FROM example;
```


Synopsis

geometry **ST_CoverageSimplify**(geometry winset geom, float8 tolerance, boolean simplifyBoundary = true);

A window function which simplifies the edges of polygons in a polygonal coverage. The simplification preserves the coverage topology. This means the simplified output polygons are consistent along shared edges, and still form a valid coverage.

The simplification uses a variant of the [Visvalingam-Whyatt algorithm](#). The *tolerance* parameter has units of distance, and is roughly equal to the square root of triangular areas to be simplified.

To simplify only the "internal" edges of the coverage (those that are shared by two polygons) set the *simplifyBoundary* parameter to false.

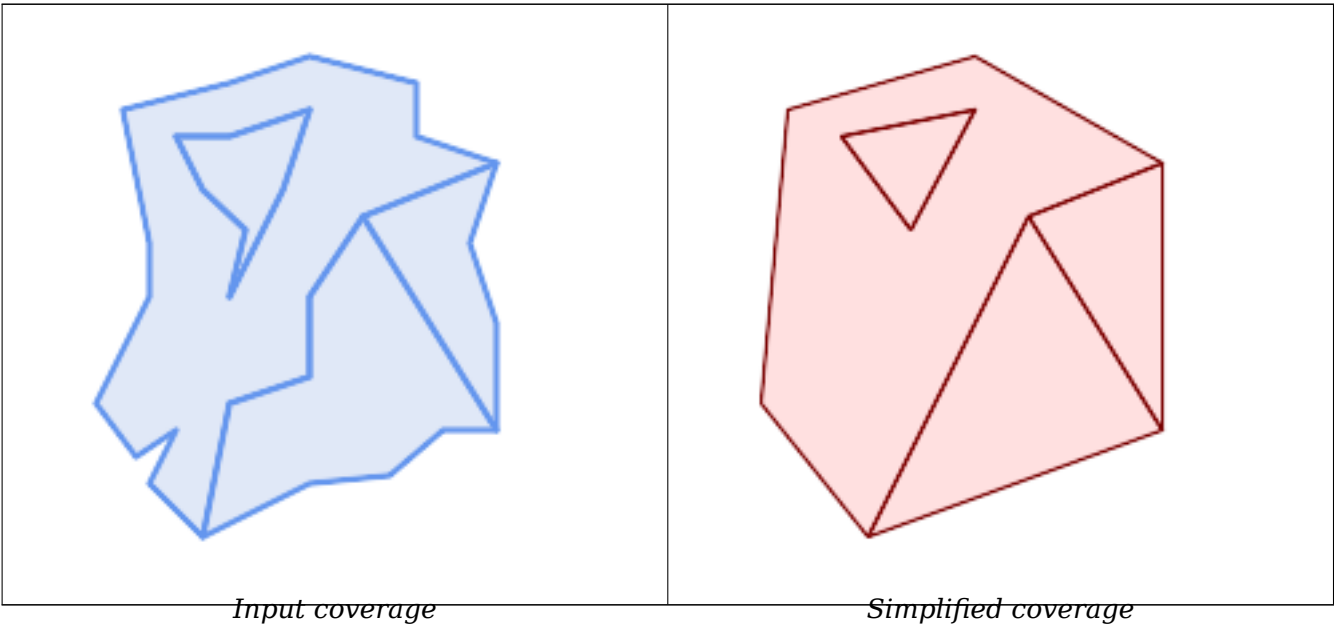


Note

If the input is not a valid coverage there may be unexpected artifacts in the output (such as boundary intersections, or separated boundaries which appeared to be shared). Use [ST_CoverageInvalidEdges](#) to determine if a coverage is valid.

Availability: 3.4.0

Requires GEOS >= 3.12.0



```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((160 150, 110 130, 90 100, 90 70, 60 60, 50 10, 30 30, 40 50, 25 40, 10 60, 30 100, 30 120, 20 170, 60 180, 90 190, 130 180, 130 160, 160 150), (40 160, 50 140, 66 125, 60 100, 80 140, 90 170, 60 160, 40 160))'::geometry),
  (2, 'POLYGON ((40 160, 60 160, 90 170, 80 140, 60 100, 66 125, 50 140, 40 160))'::geometry),
```

```
(3, 'POLYGON ((110 130, 160 50, 140 50, 120 33, 90 30, 50 10, 60 60, 90 70, 90 100, 110 130))'::geometry),
(4, 'POLYGON ((160 150, 150 120, 160 90, 160 50, 110 130, 160 150))'::geometry)
)
SELECT id, ST_AsText(ST_CoverageSimplify(geom, 30) OVER ())
FROM coverage;
```

id	st_astext
1	POLYGON ((160 150, 110 130, 50 10, 10 60, 20 170, 90 190, 160 150), (40 160, 66 125, 90 170, 40 160))
2	POLYGON ((40 160, 66 125, 90 170, 40 160))
3	POLYGON ((110 130, 160 50, 50 10, 110 130))
4	POLYGON ((160 150, 160 50, 110 130, 160 150))



[ST_CoverageInvalidEdges](#), [ST_CoverageUnion](#), [ST_CoverageClean](#)

7.15.4 ST_CoverageEdges

ST_CoverageEdges — Computes the unique edges of a polygonal coverage.

Synopsis

geometry **ST_CoverageEdges**(geometry geom, integer edgetype = 0);
geometry **ST_CoverageEdges**(geometry[] geoms, integer edgetype = 0);



Returns a MultiLineString representing the unique edges of a polygonal coverage. A polygonal coverage is a set of non-overlapping polygons where adjacent polygons have matching vertices along shared edges.

The *edgetype* parameter can be used to select which edges are returned:

- **0 (ALL)** - all unique edges (default)
- **1 (EXTERIOR)** - only exterior edges (non-shared)
- **2 (INTERIOR)** - only interior edges (shared)

Availability: 3.7.0

Requires GEOS >= 3.15.0



This example returns all unique edges of a coverage.

```
SELECT ST_AsText(ST_CoverageEdges(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0, 10 0, 10 10, 0 10, 0 0)), POLYGON((10 0, 20 0, 20 10, 10 10, 10 0)))')));
```

```
MULTILINESTRING((0 0, 10 0), (10 0, 20 0), (20 0, 20 10), (20 10, 10 10), (10 10, 0 10), (0 10, 0 0), (10 0, 10 10))
```

This example returns only the interior shared edges of a coverage.

```
SELECT ST_AsText(ST_CoverageEdges(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0, 10 0, 10 10, 0 10, 0 0)), POLYGON((10 0, 20 0, 20 10, 10 10, 10 0)))', 2));
```

```
MULTILINESTRING((10 0, 10 10))
```

☒☒

[ST_CoverageInvalidEdges](#), [ST_CoverageSimplify](#), [ST_CoverageUnion](#)

7.15.5 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.

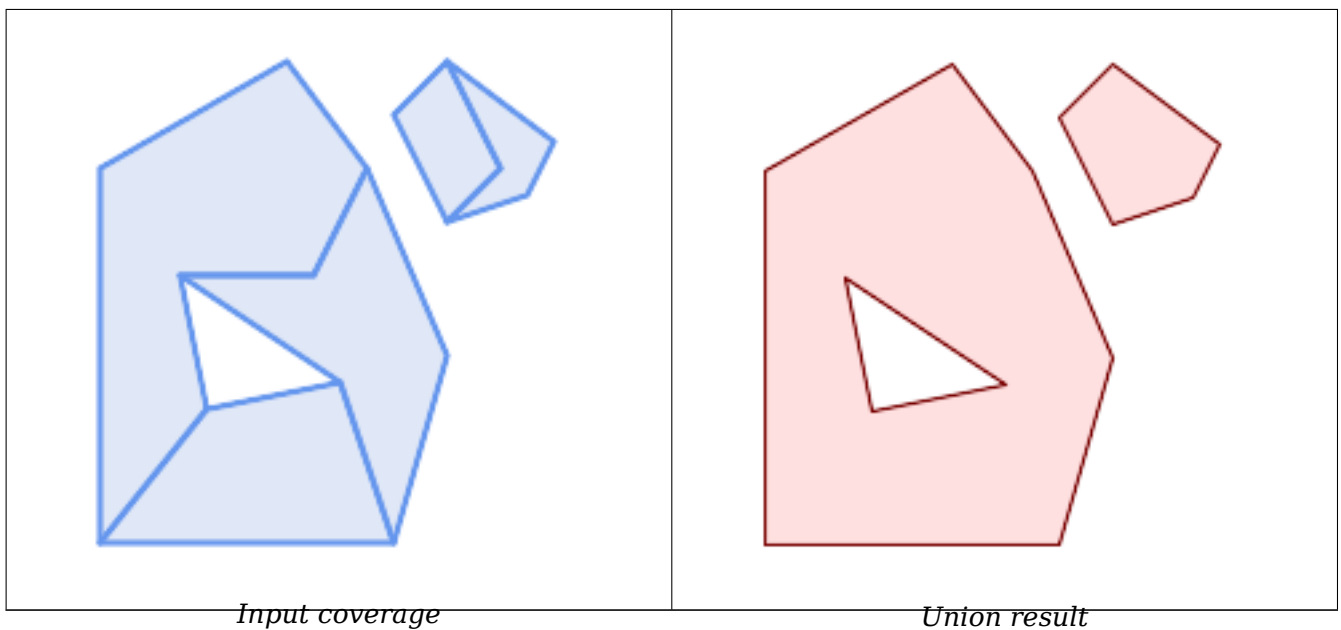


Note

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)) AS wkt
FROM coverage;
```

```
MULTIPOLYGON(((10 150,80 190,110 150,140 80,120 10,10 10,10 150)),(50 60,100 70,40 110,50 ←
  60)),((120 170,140 190,180 160,170 140,140 130,120 170)))
```

☒☒

[ST_CoverageInvalidEdges](#), [ST_CoverageSimplify](#), [ST_CoverageClean](#), [ST_Union](#)

7.16 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 引入 (curve) 几何类型。1.3.4 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.

☒☒

Rotate a 3D line 180 degrees about the z axis. Note this is long-hand for doing ST_Rotate();

```
SELECT ST_AsEWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), 0, 0, 0, 1, 0, 0, 0)) As using_affine,
       ST_AsEWKT(ST_Rotate(geom, pi())) As using_rotate
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
```

using_affine	using_rotate
LINESTRING(-1 -2 3,-1 -4 3)	LINESTRING(-1 -2 3,-1 -4 3)

(1 row)

Rotate a 3D line 180 degrees in both the x and z axis.

```
SELECT ST_AsEWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin(pi()), 0, 0, sin(pi()), cos(pi()), 0, 0, 0))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
```

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

☒☒

ST_Rotate, ST_Scale, ST_Translate, ST_TransScale

7.16.2 ST_Rotate

ST_Rotate — Rotates a geometry about an origin point.

Synopsis

```
geometry ST_Rotate(geometry geomA, float rotRadians);
geometry ST_Rotate(geometry geomA, float rotRadians, float x0, float y0);
geometry ST_Rotate(geometry geomA, float rotRadians, geometry pointOrigin);
```

☒☒

Rotates geometry rotRadians counter-clockwise about the origin point. The rotation origin can be specified either as a POINT geometry, or as x and y coordinates. If the origin is not specified, the geometry is rotated about POINT(0 0).

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒, ☒☒☒☒ TIN ☒☒☒☒☒☒☒☒☒☒.

Enhanced: 2.0.0 additional parameters for specifying the origin of rotation were added.

Availability: 1.1.2. Name changed from Rotate to ST_Rotate in 1.2.2

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

❏❏

This example rotates a geometry by 180 degrees.

```
SELECT ST_AsText(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)::geometry, pi()), 6);
```

```
LINESTRING(-50 -160,-50 -50,-100 -50)
```

This example rotates a geometry 30 degrees counter-clockwise around x=50, y=160.

```
SELECT ST_AsText(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)::geometry, pi()/6, 50, 160) ←  
, 6);
```

```
LINESTRING(50 160,105 64.737206,148.30127 89.737206)
```

This example rotates a geometry 60 degrees clockwise around its centroid.

```
SELECT ST_AsText(ST_Rotate(geom, -pi()/3, ST_Centroid(geom)), 4)  
FROM (SELECT 'LINESTRING (50 160, 50 50, 100 50)::geometry AS geom) AS foo;
```

```
LINESTRING(116.4225 130.6721,21.1597 75.6721,46.1597 32.3708)
```

❏❏

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).

❏

This example rotates a line 90 degrees around the X axis.

```
SELECT ST_AsEWKT(ST_RotateX(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
```

```
LINESTRING(1 -3 2,1 -1 1)
```

❏

[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);

❏

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

❏: 2.0.0 支持多面体表面, 支持 TIN 三角网。



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).

❏

This example rotates a line 90 degrees around the Y axis.

```
SELECT ST_AsEWKT(ST_RotateY(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
```

```
LINESTRING(3 2 -1,1 1 -1)
```

❏

[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);

⚠

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).

⚠: 2.0.0 版本中，TIN 类型不被支持。

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



Note

1.3.4 版本中，曲线（curve）类型不被支持。1.3.4 版本中，曲线类型不被支持。



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).

⚠

This example rotates a line 90 degrees around the Z axis.

```
SELECT ST_AsEWKT(ST_RotateZ(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
```

```
LINESTRING(-2 1 3,-1 1 1)
```

This example rotates a curved circle around the 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(-567 237.00000000000003,-567 231.00000000000003,-567
237.00000000000003))
```


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 $\frac{d}{dx} \left(\frac{1}{x^2} \right) = -\frac{2}{x^3}$ (curve) $y = -\frac{2}{x^3}$. **1.3.4** $\frac{d}{dx} \left(\frac{1}{x^2} \right) = -\frac{2}{x^3}$.

Availability: 1.1.0.

Version: 2.0.0, TIN

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.



This example scales X, Y, and Z values.

```
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75, 0.8));
```

```
LINESTRING(0.5 1.5 2.4,0.5 0.75 0.8)
```

This example scales X and Y values.

```
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75));
```

```
LINESTRING(0.5 1.5 3,0.5 0.75 1)
```

This example scales X, Y, Z, and M values.

```
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)'),
    ST_MakePoint(0.5, 0.75, 2, -1)));

LINESTRING(0.5 1.5 6 -4,0.5 0.75 2 -1)
```

This example scales X and Y values using a false origin.

```
SELECT ST_AsText(ST_Scale('LINESTRING(1 1, 2 2)', 'POINT(2 2)', 'POINT(1 1)::geometry'));
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 deltax);
```



Returns a new geometry whose coordinates are translated delta x,delta y,delta z units. Units are based on the units defined in spatial reference (SRID) for this geometry. M coordinates are preserved.



Note

1.3.4 $\frac{1}{x^2} = x^{-2}$ (curve) $y = -\frac{1}{x}$. 1.3.4 $\frac{1}{x^2} = x^{-2}$.

1.2.2 ☒☒☒☒☒☒☒☒☒☒☒.



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



This method supports Circular Strings and Curves.



This function supports M coordinates.



Move a point 1 degree longitude

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('POINT(-71.01 42.37)',4326),1,0)) AS ↵
      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;
```

```
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));
```

```
POINT(5 12 3)
```

Move points with a measure coordinate

```
SELECT ST_AsEWKT(ST_Translate(ST_GeomFromEWKT('POINTM(0 0 11)'), 5, 12));
```

```
POINTM(5 12 11)
```

```
SELECT ST_AsEWKT(ST_Translate(ST_GeomFromEWKT('POINT(0 0 7 11)'), 5, 12, 3));
```

```
POINT(5 12 10 11)
```

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 $\frac{1}{x^2} = x^{-2}$ (curve) $y = -\frac{1}{x}$. 1.3.4 $\frac{1}{x^2} = x^{-2}$.

Availability: 1.1.0.



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



This method supports Circular Strings and Curves.



```
SELECT ST_AsEWKT(ST_TransScale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 1, 1, 2));
```

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

This example buffers a point to approximate a circle, converts it to a curve, translates it by 1,2, and scales it by 3,4.

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

CURVEPOLYGON(CIRCULARSTRING(714 2276,696 2276,714 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.

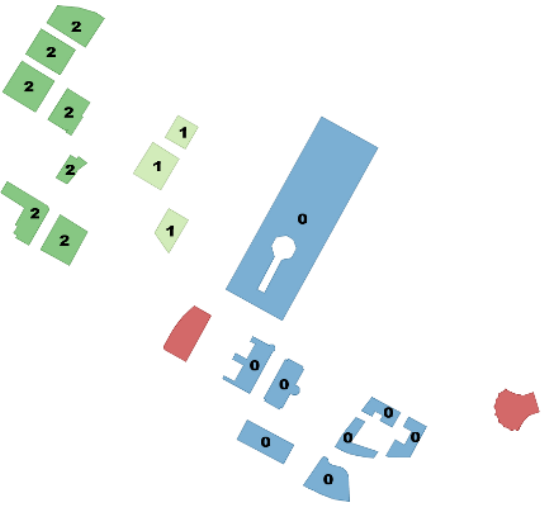
2.3.0 简体中文.



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	
1	101 Seaport	
2	District Hall	
2	One Marina Park Drive	
2	Twenty Two Liberty	
2	Vertex	
2	Vertex	
2	Watermark Seaport	
NULL	Blue Hills Bank Pavilion	
NULL	World Trade Center West	

(20 rows)

A example showing combining parcels with the same cluster number into geometry collections.

```
SELECT cid, ST_Collect(geom) AS cluster_geom, array_agg(parcel_id) AS ids_in_cluster FROM (
  SELECT parcel_id, ST_ClusterDBSCAN(geom, eps => 0.5, minpoints => 5) over () AS cid,
  geom
FROM parcels) sq
GROUP BY cid;
```

[ST_DWithin](#), [ST_ClusterKMeans](#), [ST_ClusterIntersecting](#), [ST_ClusterIntersectingWin](#), [ST_ClusterWithin](#), [ST_ClusterWithinWin](#)

7.17.2 ST_ClusterIntersecting

ST_ClusterIntersecting — Aggregate function that clusters input geometries into connected sets.

Synopsis

geometry[] **ST_ClusterIntersecting**(geometry set g);

￼￼

An aggregate function that returns an array of GeometryCollections partitioning the input geometries into connected clusters that are disjoint. Each geometry in a cluster intersects at least one other geometry in the cluster, and does not intersect any geometry in other clusters.

2.2.0 ￼￼￼￼￼￼￼￼￼.

￼￼

```
WITH testdata AS
  (SELECT unnest(
    ARRAY['LINESTRING (0 0, 1 1)::geometry',
          'LINESTRING (5 5, 4 4)::geometry',
          'LINESTRING (6 6, 7 7)::geometry',
          'LINESTRING (0 0, -1 -1)::geometry',
          'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))::geometry'] AS geom)
  )
```

```
SELECT ST_AsText(unnest(ST_ClusterIntersecting(geom))) FROM testdata;
```

```
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_ClusterRelateWin

ST_ClusterRelateWin — Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.

Synopsis

integer **ST_ClusterRelateWin**(geometry winset geom, text relate_matrix);

A window function that builds connected clusters of geometries that intersect. Geometries are added to a cluster if they share a pairwise DE9IM relationship with another member of the cluster. With this function it is possible to build a cluster of all objects that touch at boundaries, but exclude those that merely overlap.

Availability: 3.7.0

This collection of line strings would form a single cluster using **ST_ClusterIntersectingWin**, but using **ST_ClusterRelateWin** can be clustered into three groups that connect only via their end points.



Connected and overlapping linestrings

```
CREATE TABLE clusterrelate (  
  id serial,  
  geom geometry);  
  
INSERT INTO clusterrelate (geom)  
VALUES  
  ('LINESTRING(0 0,1 1)'),  
  ('LINESTRING(2 2,1 1)'),  
  ('LINESTRING(0 1,1 0)'),  
  ('LINESTRING(0 1,0 4)'),  
  ('LINESTRING(2 2,2 4)'),  
  ('LINESTRING(1.5 2.5,2.5 3.5)');  
  
SELECT id,  
  ST_AsText(geom),  
  ST_ClusterRelateWin(geom, '****0****') OVER () AS cluster  
FROM clusterrelate;
```

id	st_astext	cluster
1	LINESTRING(0 0,1 1)	0
2	LINESTRING(2 2,1 1)	0
3	LINESTRING(0 1,1 0)	1
4	LINESTRING(0 1,0 4)	1
5	LINESTRING(2 2,2 4)	0
6	LINESTRING(1.5 2.5,2.5 3.5)	2



[ST_Relate](#), [ST_ClusterIntersectingWin](#), [ST_ClusterWithinWin](#)

7.17.5 ST_MinimumSpanningTree

ST_MinimumSpanningTree — Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.

Synopsis

integer **ST_MinimumSpanningTree**(geometry winset geom);

A window function that builds connected graphs of line strings based on the Minimum Spanning Tree (MST) of the input geometries. The return value is the cluster number that the geometry argument participates in, or zero if it is not part of the minimum tree.

The **Minimum Spanning Tree** connects all geometries in the window partition such that the total length of the connecting lines is minimized. If the graph is not fully connected (e.g. infinite distance between some geometries), it produces a Minimum Spanning Forest, and each connected component is assigned a unique tree ID.

Availability: 3.7.0

Requires GEOS >= 3.15.0

```
SELECT id, ST_AsText(geom), ST_MinimumSpanningTree(geom) OVER () AS msp
FROM (VALUES
  (1, 'LINESTRING(0 0,1 0)::geometry),
  (2, 'LINESTRING(0 0,0 1)::geometry),
  (3, 'LINESTRING(1 1,0 1)::geometry),
  (4, 'LINESTRING(1 1,1 0)::geometry),
  (5, 'LINESTRING(0 0,1 1)::geometry),
  (6, 'LINESTRING(1 0,0 1)::geometry)
) AS t(id, geom);
```

id	st_astext	msp
1	LINESTRING(0 0,1 0)	1
2	LINESTRING(0 0,0 1)	1
3	LINESTRING(1 1,0 1)	1
4	LINESTRING(1 1,1 0)	0
5	LINESTRING(0 0,1 1)	0
6	LINESTRING(1 0,0 1)	0

ST_ClusterDBSCAN, **ST_ClusterKMeans**, **ST_ClusterIntersectingWin**

7.17.6 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);

国国

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

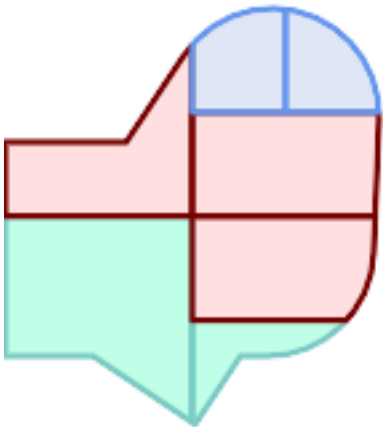
Enhanced: 3.1.0 Support for 3D geometries and weights

2.3.0 国国国国国国国国国国国国.

国国

Generate dummy set of parcels for examples:

```
CREATE TABLE parcels AS
SELECT lpad((row_number() over())::text,3,'0') As parcel_id, geom,
('{residential, commercial}'::text[])[1 + mod(row_number()OVER(),2)] As type
FROM
  ST_Subdivide(
    ST_Buffer(
      'SRID=3857;LINESTRING(40 100, 98 100, 100 150, 60 90)'::geometry,
      40, 'endcap=square'),
    12) As geom;
```



Parcels color-coded by cluster number (cid)

```
SELECT ST_ClusterKMeans(geom, 3) OVER() AS cid, parcel_id, geom
FROM parcels;
```

cid	parcel_id	geom
0	001	0103000000...
0	002	0103000000...
1	003	0103000000...
0	004	0103000000...

1		005		0103000000...
2		006		0103000000...
2		007		0103000000...

Partitioning parcel clusters by type:

```
SELECT ST_ClusterKMeans(geom, 3) over (PARTITION BY type) AS cid, parcel_id, type
FROM parcels;
```

cid		parcel_id		type
1		005		commercial
1		003		commercial
2		007		commercial
0		001		commercial
1		004		residential
0		002		residential
2		006		residential

Example: Clustering a preaggregated planetary-scale data population dataset using 3D clusering and weighting. Identify at least 20 regions based on **Kontur Population Data** that do not span more than 3000 km from their center:

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



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

☒☒

ST_ClusterDBSCAN, **ST_ClusterIntersectingWin**, **ST_ClusterWithinWin**, **ST_ClusterIntersecting**, **ST_ClusterST_Subdivide**, **ST_Force3D**, **ST_Force4D**,

7.17.7 ST_ClusterWithin

ST_ClusterWithin — Aggregate function that clusters geometries by separation distance.

Synopsis

geometry[] **ST_ClusterWithin**(geometry set g, float8 distance);

⊠

An aggregate function that returns an array of GeometryCollections, where each collection is a cluster containing some input geometries. Clustering partitions the input geometries into sets in which each geometry is within the specified *distance* of at least one other geometry in the same cluster. Distances are Cartesian distances in the units of the SRID.

ST_ClusterWithin is equivalent to running **ST_ClusterDBSCAN** with `minpoints => 0`.

2.2.0 ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠.



This method supports Circular Strings and Curves.

⊠

```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)::geometry',
    'LINESTRING (5 5, 4 4)::geometry',
    'LINESTRING (6 6, 7 7)::geometry',
    'LINESTRING (0 0, -1 -1)::geometry',
    'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))::geometry']) AS geom)
```

```
SELECT ST_AsText(unnest(ST_ClusterWithin(geom, 1.4))) FROM testdata;
```

```
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**

7.17.8 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.

Compatibility: 2.0.0 **Supported:** **Yes**, **Yes** TIN **Supported:** **Yes**.

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

例

```
SELECT Box2D(ST_GeomFromText('LINESTRING(1 2, 3 4, 5 6)'));
```

```
BOX(1 2,5 6)
```

```
SELECT Box2D(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'));
```

```
BOX(220186.99512189245 150406,220288.24878054656 150506.12682932706)
```

例

Box3D, ST_GeomFromText

7.18.2 Box3D

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

Synopsis

box3d **Box3D**(geometry geom);

例

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

例: 2.0.0 版本, 支持 TIN 格式。

- ✓ 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 Box3D(ST_GeomFromEWKT('LINESTRING(1 2 3, 3 4 5, 5 6 5)'));
```

```
BOX3D(1 2 3,5 6 5)
```

```
SELECT Box3D(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 1,220227 150406 1)'));
```

```
BOX3D(220186.99512189245 150406 1,220288.24878054656 150506.12682932706 1)
```

国国

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);
```

国国

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 国国国国国国国国国国国国.

Changed: 2.1.0. Up to 2.0.x this was called ST_Estimated_Extent.



This method supports Circular Strings and Curves.

国国

```
SELECT ST_EstimatedExtent('ny', 'edges', 'geom');
BOX(-8877653 4912316,-8010225.5 5589284)
```

```
SELECT ST_EstimatedExtent('feature_poly', 'geom');
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);
```

```
BOX(2312882 110666,2312990 110724)
```

This example expands a 3D box by 10 meters.

```
SELECT ST_Expand(CAST('BOX3D(778783 2951741 1,794875 2970042.61545891 10)' AS box3d),10)
```

```
BOX3D(778773 2951731 -9,794885 2970052.61545891 20)
```

This example renders the text representation of a 10-meter expanded 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.

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒, ☒☒☒☒ TIN ☒☒☒☒☒☒☒☒☒☒.



This function supports Polyhedral surfaces.



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



Note
Examples below use Massachusetts State Plane ft (SRID=2249)

```
SELECT ST_Extent(geom) AS bextent FROM sometable;
```

```
B0X(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
B0X(778783.5625 2951741.25,794875.8125 2970042.75)	A
B0X(751315.8125 2919164.75,765202.6875 2935417.25)	B
B0X(739651.875 2917394.75,756688.375 2935866)	C

This example converts the extent back into a geometry and renders its extended text representation.

```
SELECT ST_SetSRID(ST_Extent(geom),2249) AS bextent FROM sometable;
```

```
SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,
794875.8125 2908247.25,739651.875 2908247.25))
```

[ST_EstimatedExtent](#), [ST_3DExtent](#), [ST_SetSRID](#)

7.18.6 ST_3DExtent

ST_3DExtent — Aggregate function that returns the 3D bounding box of geometries.

Synopsis

```
box3d ST_3DExtent(geometry set geomfield);
```

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.

￼￼￼: 2.0.0 ￼￼￼￼￼￼￼, ￼￼￼ TIN ￼￼￼￼￼￼￼￼.

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).

￼￼

```
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;
```

```
BOX3D(1 1 0,3 2 2)
```

This example returns the extent of several elevated CircularStrings.

```
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_Translate(ST_Force3DZ(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;
```

```
BOX3D(0 0 0,4 3 2)
```

￼￼

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))
```

```
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)');
```

```
4
```

```
SELECT ST_XMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
```

```
5
```

```
SELECT ST_XMax(CAST('BOX(-3 2, 3 4)' AS box2d));
```

```
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)'));
```

```
220288.248780547
```



ST_XMin, ST_YMin, ST_YMax, ST_ZMin, ST_ZMax, ST_MMin, ST_MMax

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);

返回

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.

返回

```
SELECT ST_XMin('BOX3D(1 2 3, 4 5 6)');
```

```
1
```

```
SELECT ST_XMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
```

```
1
```

```
SELECT ST_XMin(CAST('BOX(-3 2, 3 4)' AS box2d));
```

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

```
220186.995121892
```

返回

ST_XMax, **ST_YMin**, **ST_YMax**, **ST_ZMin**, **ST_ZMax**, **ST_MMin**, **ST_MMax**

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)');
```

```
5
```

```
SELECT ST_YMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
```

```
6
```

```
SELECT ST_YMax(CAST('BOX(-3 2, 3 4)' AS box2d));
```

```
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)'));
```

```
150506.126829327
```



ST_XMin, ST_XMax, ST_YMin, ST_ZMin, ST_ZMax, ST_MMin, ST_MMax

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);

返回

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.

返回

```
SELECT ST_YMin('BOX3D(1 2 3, 4 5 6)');
```

```
2
```

```
SELECT ST_YMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
```

```
3
```

```
SELECT ST_YMin(CAST('BOX(-3 2, 3 4)' AS box2d));
```

```
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)'));
```

```
150406
```

返回

[ST_GeomFromEWKT](#), [ST_XMin](#), [ST_XMax](#), [ST_YMax](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMin](#), [ST_MMax](#)

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)');
```

```
6
```

```
SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
```

```
7
```

```
SELECT ST_ZMax('BOX3D(-3 2 1, 3 4 1)');
```

```
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)'));
```

```
3
```



[ST_GeomFromEWKT](#), [ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMin](#), [ST_MMin](#), [ST_MMax](#)

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);

返回

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.

返回

```
SELECT ST_ZMin('BOX3D(1 2 3, 4 5 6)');
```

```
3
```

```
SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
```

```
4
```

```
SELECT ST_ZMin('BOX3D(-3 2 1, 3 4 1)');
```

```
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)'));
```

```
1
```

返回

[ST_GeomFromEWKT](#), [ST_GeomFromText](#), [ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMax](#), [ST_MMin](#), [ST_MMax](#)

7.18.15 ST_XSize

ST_XSize — Returns the X size of a 2D or 3D bounding box or a geometry.

Synopsis

```
float ST_XSize(box3d aGeomorBox2DorBox3D);
```

⊠

Returns the X size of a 2D or 3D bounding box or a geometry, computed as `ST_XMax(aGeomorBox2DorBox3D) - ST_XMin(aGeomorBox2DorBox3D)`.



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.

Availability: 3.7.0

⊠

```
SELECT ST_XSize('BOX3D(1 2 3, 4 5 6)');
```

```
3
```

```
SELECT ST_XSize(ST_GeomFromText('LINESTRING(1 3, 5 6)'));
```

```
4
```

⊠

ST_XMin, **ST_XMax**, **ST_YSize**, **ST_ZSize**, **ST_MSize**

7.18.16 ST_YSize

ST_YSize — Returns the Y size of a 2D or 3D bounding box or a geometry.

Synopsis

```
float ST_YSize(box3d aGeomorBox2DorBox3D);
```

￼￼

Returns the Y size of a 2D or 3D bounding box or a geometry, computed as `ST_YMax(aGeomorBox2DorBox3D)` - `ST_YMin(aGeomorBox2DorBox3D)`.

**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.

Availability: 3.7.0

￼￼

```
SELECT ST_YSize('BOX3D(1 2 3, 4 5 6)');
```

```
3
```

```
SELECT ST_YSize(ST_GeomFromText('LINESTRING(1 3, 5 6)'));
```

```
3
```

￼￼

[ST_YMin](#), [ST_YMax](#), [ST_XSize](#), [ST_ZSize](#), [ST_MSize](#)

7.18.17 ST_ZSize

`ST_ZSize` — Returns the Z size of a 2D or 3D bounding box or a geometry.

Synopsis

```
float ST_ZSize(box3d aGeomorBox2DorBox3D);
```

￼￼

Returns the Z size of a 2D or 3D bounding box or a geometry, computed as `ST_ZMax(aGeomorBox2DorBox3D)` - `ST_ZMin(aGeomorBox2DorBox3D)`.

**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.

Availability: 3.7.0

```
SELECT ST_ZSize('BOX3D(1 2 3, 4 5 6)');
```

```
3
```

```
SELECT ST_ZSize(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
```

```
3
```

[ST_ZMin](#), [ST_ZMax](#), [ST_XSize](#), [ST_YSize](#), [ST_MSize](#)

7.18.18 ST_MMin

ST_MMin — Returns the M minima of a geometry.

Synopsis

float **ST_MMin**(geometry geom);

Returns the M minima of a geometry, or null if the geometry lacks M values.

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

```
SELECT ST_MMin('POINT M (1 2 3)');
```

```
3
```

[ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMax](#), [ST_MSize](#)

7.18.19 ST_MMax

ST_MMax — Returns the M maxima of a geometry.

Synopsis

float **ST_MMax**(geometry geom);



Returns the M maxima of a geometry, or null if the geometry lacks M values.



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



This method supports Circular Strings and Curves.



```
SELECT ST_MMax('POINT M (1 2 3)');
```

```
3
```



[ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMin](#), [ST_MSize](#)

7.18.20 ST_MSize

ST_MSize — Returns the M size of a geometry.

Synopsis

float **ST_MSize**(geometry geom);



Returns the M size of a geometry, computed as `ST_MMax(geom) - ST_MMin(geom)`, or null if the geometry lacks M values.



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



This method supports Circular Strings and Curves.

Availability: 3.7.0

例

```
SELECT ST_MSize('LINESTRING M (1 2 3, 4 6 8)');
```

```
5
```

例

ST_MMin, **ST_MMax**, **ST_XSize**, **ST_YSize**, **ST_ZSize**

7.19 线性参考 (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);

例

返回沿给定线段的指定比例位置的点。比例范围从 0 到 1。Float8 参数指定比例位置。

对于 2D 几何体，该函数返回与 **ST_LineLocatePoint** 相同的点。



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 索引（如果有）将被忽略。比例范围从 0.0 到 1.0。

0.8.2 版本中，1.1.1 版本中 Z 和 M 索引将被忽略。

兼容性：2.1.0 版本，在 2.0.x 版本中，**ST_Line_Interpolate_Point** 函数。



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

图



20% 点 (0.20) 沿线的点

This example returns the point 20 percent along a line.

```
SELECT ST_AsEWKT(ST_LineInterpolatePoint(
  'LINESTRING(25 50, 100 125, 150 190)',
  0.2));
```

```
POINT(51.5974135047432 76.5974135047432)
```

The mid-point of a 3D line:

```
SELECT ST_AsEWKT(ST_LineInterpolatePoint(
  'LINESTRING(1 2 3, 4 5 6, 6 7 8)',
  0.5));
```

```
POINT(3.5 4.5 5.5)
```

The closest point on a line to a point:

```
SELECT ST_AsText(ST_LineInterpolatePoint(
  line.geom,
  ST_LineLocatePoint(line.geom, 'POINT(4 3)')))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As geom) AS line;
```

```
POINT(3 4)
```

图

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

7.19.2 ST_3DLineInterpolatePoint

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

Synopsis

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

Notes

Computes points in 2D and then interpolates the values for Z and M. Float8 0 to 1.



Note

ST_LineInterpolatePoint computes points in 2D and then interpolates the values for Z and M, while this function computes points in 3D and only interpolates the M value.

2.1.0



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

Notes

Return point 20% along 3D line

```
SELECT ST_AsText(ST_3DLineInterpolatePoint(
  'LINESTRING(25 50 70, 100 125 90, 150 190 200)',
  0.20));
```

```
POINT Z (59.06758929108215 84.06758929108216 79.0846904776219)
```

Notes

ST_LineInterpolatePoint, **ST_LineInterpolatePoint**, **ST_LineMerge**

7.19.3 ST_LineInterpolatePoints

ST_LineInterpolatePoints — Returns points interpolated along a line at a fractional interval.

Synopsis

geometry **ST_LineInterpolatePoints**(geometry a_linestring, float8 a_fraction, boolean repeat);
 geography **ST_LineInterpolatePoints**(geography a_linestring, float8 a_fraction, boolean use_spheroid = true, boolean repeat = true);

Notes

Returns one or more points interpolated along a line at a fractional interval. The first argument must be a LINESTRING. The second argument is a float8 between 0 and 1 representing the spacing between the points as a fraction of line length. If the third argument is false, at most one point will be constructed (which is equivalent to **ST_LineInterpolatePoint**.)

If the result has zero or one points, it is returned as a POINT. If it has two or more points, it is returned as a MULTIPOINT.

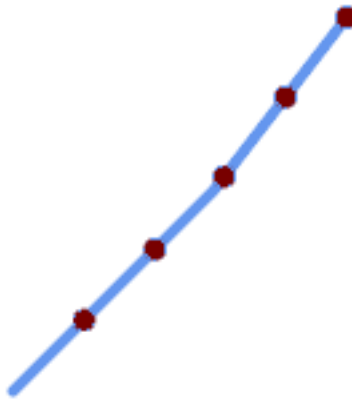
Availability: 2.5.0



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



This function supports M coordinates.



A LineString with points interpolated every 20%

This example returns points every 20 percent along a two-dimensional 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))
```



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);
```

图例

图例显示了如何从 2 个图例中生成 0 到 1 的浮点图例。

图例显示了 `(ST_LineInterpolatePoint)` 图例和 `(ST_LineSubstring)` 图例。

图例显示了图例。

1.1.0 图例显示了图例。

图例: 2.1.0 图例, 图 2.0.x 图例 `ST_Line_Locate_Point` 图例。

图例

This example approximates the street number of a house by locating the closest point on a street line. The subquery generates sample house and street data, and `ST_DWithin` excludes houses that are too far from 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

This example finds the 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;
```

POINT(3 4)

图例

`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);
```

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. For geometry inputs, the fractions are measured in 2D line length. The Z and M values are interpolated for added endpoints if present.

' ' ' ' ST_LineInterpolatePoint



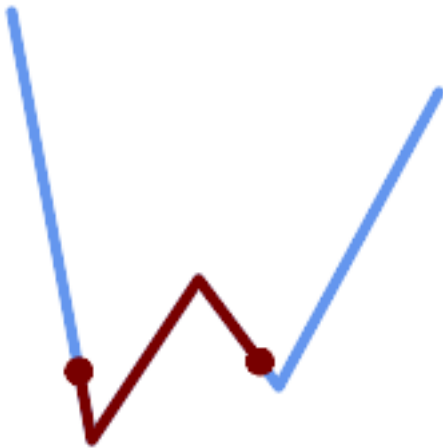
Note
This only works with LINESTRINGs. To use on contiguous MULTILINESTRINGs first join them with ST_LineMerge.



Note
1.1.1 M Z () . .

Enhanced: 3.4.0 - Support for geography was introduced.
: 2.1.0 , 2.0.x ST_Line_Substring .
1.1.0 . 1.1.1 Z M .

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



1/3 (0.333, 0.666)

```
SELECT ST_AsText(ST_LineSubstring(
  'LINESTRING (20 180, 50 20, 90 80, 120 40, 180 150)',
  0.333,
  0.666));
```

```
LINESTRING (45.17311810399485 45.74337011202746, 50 20, 90 80, 112.97593050157862
49.36542599789519) ↵
```

If start and end locations are the same, the result is a POINT.

```
SELECT ST_AsText(ST_LineSubstring(
  'LINESTRING(25 50, 100 125, 150 190)',
  0.333,
  0.333));
```

```
POINT(69.28469348539744 94.28469348539744)
```

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(-103.911641 38.931128,-87.941787 41.831072)		LINESTRING(-102.530462 36.819064,-86.817324 39.585927) ←

For geometry inputs, the fractional locations are based on the 2D length of the line, even when the input has Z values. The resulting Z values are interpolated along the selected 2D location.

```
WITH data AS (  
    SELECT 'LINESTRING Z (0 0 0, 0 2 5, 0 10 10)::geometry' AS geom  
)  
SELECT ST_Length(geom) AS length_2d,  
       ST_3DLength(geom) AS length_3d,  
       ST_AsText(ST_LineSubstring(geom, 0, 0.5)) AS substring  
FROM data;
```

length_2d		length_3d		substring
-----+-----				
10		14.819145939191106		LINESTRING Z (0 0 0,0 2 5,0 5 6.875)

ST_Length, ST_LineExtend, ST_LineInterpolatePoint, ST_LineMerge

7.19.6 ST_LocateAlong

ST_LocateAlong — Returns the point(s) on a geometry that match a measure value.

Synopsis

geometry **ST_LocateAlong**(geometry geom_with_measure, float8 measure, float8 offset = 0);

Returns the location(s) along a measured geometry that have the given measure values. The result is a Point or MultiPoint. Polygonal inputs are not supported.

If offset is provided, the result is offset to the left or right of the input line by the specified distance. A positive offset will be to the left, and a negative one to the right.



Note
Use this function only for linear geometries with an M component

The semantic is specified by the *ISO/IEC 13249-3 SQL/MM Spatial* standard.

1.1.0 ST_LocateAlong_Measure

2.0.0 ST_LocateAlong_Measure. SQL/MM IEC 13249-3: 5.1.13

- ✓ This function supports M coordinates.
- ✓ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.13

例

```
SELECT ST_AsText(ST_LocateAlong(
  'MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))'::geometry,
  3));
```

```
MULTIPOINT M ((1 2 3),(9 4 3),(1 2 3))
```

例

[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);

例

此函数返回指定几何体中位于指定测量范围内的部分。如果指定了偏移量，则返回的几何体将从指定的测量值开始。如果指定了测量范围，则返回的几何体将位于指定的测量值之间。如果指定了测量范围，则返回的几何体将位于指定的测量值之间。

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。

2.0.0 添加 `ST_Locate_Along_Measure` 函数。此函数返回指定几何体中位于指定测量范围内的部分。如果指定了偏移量，则返回的几何体将从指定的测量值开始。如果指定了测量范围，则返回的几何体将位于指定的测量值之间。如果指定了测量范围，则返回的几何体将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.

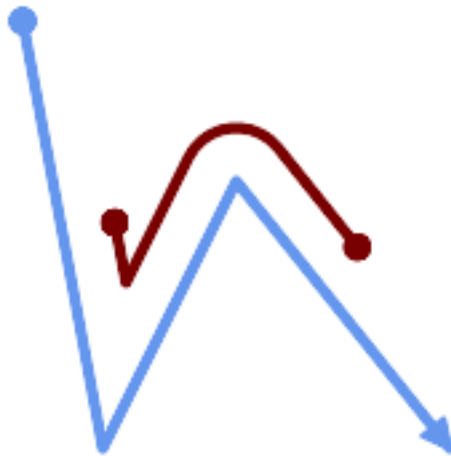


This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

例

```
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  ←
82.12248075654186,82.11145618000168 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 简体中文.

Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE.



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

例

```
SELECT ST_AsText(ST_LocateBetweenElevations(
  'LINESTRING(1 2 3, 4 5 6)::geometry',
  2,
  4));
```

```
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;
```

```
GEOMETRYCOLLECTION Z (POINT Z (1 2 6),LINESTRING Z (6.1 7.1 6,7 8 9))
```

例

[ST_Dump](#), [ST_LocateAlong](#), [ST_LocateBetween](#)

7.19.9 ST_InterpolatePoint

`ST_InterpolatePoint` — 返回沿线性几何体 (M 分量) 的插值点。

Synopsis

`float8 ST_InterpolatePoint(geometry linear_geom_with_measure, geometry point);`

例

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 简体中文.



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

❏❏

```
SELECT ST_InterpolatePoint(
  'LINESTRING M (0 0 0, 10 0 20)',
  'POINT(5 5)');
```

10

❏❏

[ST_AddMeasure](#), [ST_LocateAlong](#), [ST_LocateBetween](#)

7.19.10 ST_3DInterpolatePoint

ST_3DInterpolatePoint — Returns the interpolated measure of a geometry closest to a point in 3D.

Synopsis

float8 **ST_3DInterpolatePoint**(geometry linear_geom_with_measure, geometry point);

❏❏

Returns the interpolated measure value of a linear ZM geometry at the location closest to the given point, using 3D (XYZ) distance for the projection. Use this function when the geometry has significant Z variation, such as flight trajectories, where [ST_InterpolatePoint](#) would give incorrect results by ignoring the Z dimension.



Note

The line must have both Z and M dimensions. The point should have a Z dimension.

Availability: 3.7.0



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

❏❏

This example interpolates on a line rising diagonally in Z, using a point at the 3D midpoint.

```
SELECT ST_3DInterpolatePoint(
  'LINESTRING ZM (0 0 0 0, 10 0 10 20)',
  'POINT Z (5 0 5)');
```

10

This example uses a vertical line with zero XY extent, where 3D projection works but 2D projection does not.

```
SELECT ST_3DInterpolatePoint(
  'LINESTRING ZM (0 0 0 0, 0 0 10 100)',
  'POINT Z (5 5 5)');
```

50

ST_InterpolatePoint, ST_AddMeasure, ST_LocateAlong, ST_LocateBetween

7.19.11 ST_AddMeasure

ST_AddMeasure — Interpolates measures along a linear geometry.

Synopsis

```
geometry ST_AddMeasure(geometry geom_mline, float8 measure_start, float8 measure_end);
```



本報告係根據本會所屬之「國家安全情報中心」所蒐集之資料，經本會整理、分析、彙編而成。本報告之內容，係根據本會所屬之「國家安全情報中心」所蒐集之資料，經本會整理、分析、彙編而成。本報告之內容，係根據本會所屬之「國家安全情報中心」所蒐集之資料，經本會整理、分析、彙編而成。

1.5.0 ☒☒☒☒☒☒☒☒☒☒☒.

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



```
SELECT ST_AsText(ST_AddMeasure(
    ST_GeomFromEWKT('LINESTRING(1 0, 2 0, 4 0)'),
    1,
    4)) As ewelev;
```

```
LINESTRING M (1 0 1,2 0 2,4 0 4)
```

```
SELECT ST_AsText(ST_AddMeasure(
    ST_GeomFromEWKT('LINESTRING(1 0 4, 2 0 4, 4 0 4)'),
    10,
    40)) As ewelev;
```

```
LINESTRING ZM (1 0 4 10,2 0 4 20,4 0 4 40)
```

```
SELECT ST_AsText(ST_AddMeasure(
    ST_GeomFromEWKT('LINESTRINGM(1 0 4, 2 0 4, 4 0 4)'),
    10,
    40)) As ewelev;
```

```
LINESTRING M (1 0 10,2 0 20,4 0 40)
```

```
SELECT ST_AsText(ST_AddMeasure(
    ST_GeomFromEWKT('MULTILINESTRINGM((1 0 4, 2 0 4, 4 0 4),(1 0 4, 2 0 4, 4 0 4))'),
    10,
    70)) As ewelev;
```

```
MULTILINESTRING M ((1 0 10,2 0 20,4 0 40),(1 0 40,2 0 50,4 0 70))
```

7.20 Trajectory Functions

7.20.1 ST_IsValidTrajectory

`ST_IsValidTrajectory` — Tests if the geometry is a valid trajectory.

Synopsis

boolean **ST_IsValidTrajectory**(geometry line);

⚠️

Tests if a geometry encodes a valid trajectory. A valid trajectory is represented as a `LINESTRING` with measures (M values). The measure values must increase from each vertex to the next.

Valid trajectories are expected as input to spatio-temporal functions like [ST_ClosestPointOfApproach](#) 2.2.0 简体中文.



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

⚠️

This example constructs a valid trajectory.

```
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0, 0, 1),
    ST_MakePointM(0, 1, 2)));
```

```
t
```

This example constructs 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)
f
```

⚠️

[ST_ClosestPointOfApproach](#)

7.20.2 ST_ClosestPointOfApproach

`ST_ClosestPointOfApproach` — Returns a measure at the closest point of approach of two trajectories.

Synopsis

float8 **ST_ClosestPointOfApproach**(geometry track1, geometry track2);



Returns the smallest measure at which points interpolated along the given trajectories are the least distance apart.

Inputs must be valid trajectories as checked by `ST_IsValidTrajectory`. Null is returned if the trajectories do not overlap in their M ranges.

To obtain the actual points at the computed measure use `ST_LocateAlong`.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.



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



```
-- Return the time in which two objects moving between 10:00 and 11:00
-- are closest to each other and their distance at that point
WITH inp AS ( SELECT
    ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)')::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)')::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) 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) AT TIME ZONE 'UTC' t,
       ST_3DDistance(pa,pb) distance,
       ST_AsText(pa, 2) AS pa, ST_AsText(pb, 2) AS pb
FROM points, cpa;
```

t	distance pb	pa	↔
2015-05-26 10:45:31.034483	1.9652147377620688	POINT ZM (7.59 0 3.79 1432637131.03)	↔
	POINT ZM (9.1 1.24 3.93 1432637131.03)		



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 ☒☒☒☒☒☒☒☒☒☒.



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 timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry',
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) b
)
SELECT ST_DistanceCPA(a,b) distance FROM inp;
```

1.9652147377620688

☒☒

[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 timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry',
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) b
)
SELECT ST_CPAWithin(a,b,2), ST_DistanceCPA(a,b) distance FROM inp;
```

st_cpawithin	distance
t	1.9652147377620688

❏

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, and postgis_topology.

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
```

☒☒

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="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
```

☒☒

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();
```

```
3.12.0dev-CAPI-1.18.0
```

☒☒

[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();
```

```
3.12.0
```

☒☒

[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();
```

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```

￼￼

[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();
```

```
2.9.10
```

￼￼

[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();
```

```
0.17
```

￼￼

[PostGIS_Full_Version](#), [PostGIS_Lib_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_Versio](#)

7.21.8 PostGIS_Lib_Build_Date

`PostGIS_Lib_Build_Date` — Returns build date of the PostGIS library.

Synopsis

text **PostGIS_Lib_Build_Date()**;

￼￼

Returns build date of the PostGIS library.

￼￼

```
SELECT PostGIS_Lib_Build_Date();
```

```
2023-06-22 03:56:11
```

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();
```

```
3.4.0dev
```

☐☐

[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();
```

```
7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org USER_WRITABLE_DIRECTORY=/tmp/ ↔
proj DATABASE_PATH=/usr/share/proj/proj.db
```

☐☐

[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()**;

PROJ

Returns the version number of the PROJ library, or against which PostGIS was built.

Availability: 3.5.0

SQL

```
SELECT PostGIS_PROJ_Compiled_Version();
```

9.1.1

See Also

[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()**;

WAGYU

Returns the version number of the internal Wagyu library, or NULL if Wagyu support is not enabled.

SQL

```
SELECT PostGIS_Wagyu_Version();
```

0.5.0 (Internal)

See Also

[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();
```

```
2023-06-22 03:56:11
```

[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();
```

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```



[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();
```

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```



[PostGIS_Full_Version](#), [PostGIS_Scripts_Installed](#), [PostGIS_Lib_Version](#)

7.21.16 PostGIS_Version

`PostGIS_Version` — Returns PostGIS version number and compile-time options.

Synopsis

text **`PostGIS_Version()`**;



Returns PostGIS version number and compile-time options.

❏

```
SELECT PostGIS_Version();
```

```
3.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
```

❏

`PostGIS_Full_Version`, `PostGIS_GEOS_Version`, `PostGIS_Lib_Version`, `PostGIS_LibXML_Version`, `PostGIS_PROJ_Version`

7.22 PostGIS GUC(Grand Unified Custom Variable)

7.22.1 postgis.gdal_datapath

`postgis.gdal datapath` — GDAL 的 GDAL_DATA 环境变量路径。如果未设置，则使用 GDAL_DATA 环境变量。

❏

GDAL 的 GDAL_DATA 环境变量是 PostgreSQL GUC 的一部分。 `postgis.gdal_datapath` 是 GDAL 的 GDAL_DATA 环境变量。

GDAL 的 GDAL_DATA 环境变量 (hard-coded) 是 GDAL 的 GDAL_DATA 环境变量。 GDAL 的 GDAL_DATA 环境变量是 GDAL 的 GDAL_DATA 环境变量。



Note

PostgreSQL 的 `postgresql.conf` 文件是 GDAL 的 GDAL_DATA 环境变量。 GDAL 的 GDAL_DATA 环境变量是 GDAL 的 GDAL_DATA 环境变量。

2.2.0 的 GDAL 的 GDAL_DATA 环境变量。



Note

GDAL 的 `GDAL_DATA` 环境变量是 GDAL 的 GDAL_DATA 环境变量。

❏

`postgis.gdal_datapath` 的 GDAL 的 GDAL_DATA 环境变量。

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';
SET postgis.gdal_datapath TO default;
```

GDAL 的 GDAL_DATA 环境变量是 GDAL 的 GDAL_DATA 环境变量。

```
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 驱动程序列表。
GDAL 使用 GDAL_SKIP 驱动程序。

☐☐

PostGIS 使用 GDAL 驱动程序。 GDAL 使用 GDAL_SKIP 驱动程序。
PostgreSQL 使用 postgresql.conf 驱动程序。 驱动程序列表

PostgreSQL 使用 POSTGIS_GDAL_ENABLED_DRIVERS
☐☐☐ (pass) ☐☐ postgis.gdal_enabled_drivers 驱动程序。

GDAL 使用 GDAL_SKIP 驱动程序。 驱动程序列表
☐ GDAL 驱动程序 驱动程序。 驱动程序列表。

Note

postgis.gdal_enabled_drivers 驱动程序。 驱动程序列表。

Note!

- DISABLE_ALL 使用 GDAL 驱动程序。 DISABLE_ALL 使用, postgis.gdal_enabled_drivers 驱动程序。
- ENABLE_ALL 使用 GDAL 驱动程序。
- VSICURL 使用 GDAL 使用 /vsicurl/ 驱动程序。

postgis.gdal_enabled_drivers 使用 DISABLE_ALL 使用, DB 使用, ST_FromGDALRaster(), ST_AsGDALRaster(), ST_AsTIFF(), ST_AsJPEG() 使用 ST_AsPNG() 使用。

Note!

Note

PostGIS 使用, postgis.gdal_enabled_drivers 使用 DISABLE_ALL 使用。

Note!

Note

GDAL_SKIP 使用 GDAL 使用 Configuration Options 使用。

2.2.0 使用。

☐☐

Set and reset postgis.gdal_enabled_drivers for the current session.

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.gdal_enabled_drivers = default;
```

Set specific drivers for all new connections to a database.

```
ALTER DATABASE mygisdb SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

Set the default for the whole database cluster to enable all drivers. This requires superuser access. Also note that database, session, and user settings override this.

This example writes the setting to postgres.auto.conf.

```
ALTER SYSTEM SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

Reload the PostgreSQL configuration so the changed setting is applied.

```
SELECT pg_reload_conf();
```

☐☐

[ST_FromGDALRaster](#), [ST_AsGDALRaster](#), [ST_AsTIFF](#), [ST_AsPNG](#), [ST_AsJPEG](#), [postgis.enable_outdb_raster](#)

7.22.3 postgis.enable_outdb_rasters

postgis.enable_outdb_rasters — DB 是否启用外部栅格数据。

☐☐

DB 是否启用外部栅格数据。PostgreSQL 通过 postgresql.conf 中的 `postgis.enable_outdb_rasters` 配置项来控制。默认值为 `off`。

PostgreSQL 通过 `0` 禁用外部栅格数据，通过 `1` 启用外部栅格数据。在 PostgreSQL 启动时，可以通过 `POSTGIS_ENABLE_OUTDB_RASTERS` 环境变量（pass）来设置 `postgis.enable_outdb_rasters` 的值。



Note

`postgis.enable_outdb_rasters` 是一个配置项，GUC `postgis.enable_outdb_rasters` 是一个配置项。



Note

在 PostGIS 3.7.0 中，`postgis.enable_outdb_rasters` 是一个配置项。

2.2.0 是否启用外部栅格数据。

￼￼

Set and reset `postgis.enable_outdb_rasters` for the current session.

```
SET postgis.enable_outdb_rasters TO True;
SET postgis.enable_outdb_rasters = default;
SET postgis.enable_outdb_rasters = True;
SET postgis.enable_outdb_rasters = False;
```

Set the default for all new connections to a database.

```
ALTER DATABASE gisdb SET postgis.enable_outdb_rasters = true;
```

Set the default for the whole database cluster. This requires superuser access. Also note that database, session, and user settings override this.

This example writes the setting to `postgres.auto.conf`.

```
ALTER SYSTEM SET postgis.enable_outdb_rasters = true;
```

Reload the PostgreSQL configuration so the changed setting is applied.

```
SELECT pg_reload_conf();
```

￼￼

`postgis.gdal_enabled_drivers` `postgis.gdal_vsi_options`

7.22.4 postgis.gdal_vsi_options

`postgis.gdal_vsi_options` — DB ￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼.

￼￼

A string configuration to set options used when working with an out-db raster. [Configuration options](#) control things like how much space GDAL allocates to local data cache, whether to read overviews, and what access keys to use for remote out-db data sources.

Availability: 3.2.0

￼￼

`postgis.enable_outdb_rasters` ￼￼￼￼￼￼￼￼￼￼￼.

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxx 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=xxxxxxxxxxxxxxxx ↵
AWS_SECRET_ACCESS_KEY=yyyyyyyyyyyyyyyyyyyyyyyyyyyyyy';
```

￼￼

`postgis.enable_outdb_rasters` `postgis.gdal_enabled_drivers`

7.22.5 postgis.gdal_cpl_debug

`postgis.gdal_cpl_debug` — A boolean configuration to turn logging of GDAL debug messages on and off.



By default, GDAL logging is printed to stderr, and lower level debug messages are not printed at all. Turning this GUC to true will cause GDAL logging to be sent into the PostgreSQL logging stream, so you can see more or less of it by altering the `client_min_message` PostgreSQL GUC.

Availability: 3.6.0



```
postgis.enable_outdb_rasters postgis.gdal enabled drivers
```

7.23 Troubleshooting Functions

7.23.1 PostGIS_AddBBox

PostGIS_AddBBox — ☒☒☒☒☒☒☒☒☒☒☒☒☒.

Synopsis

```
geometry PostGIS_AddBBox(geometry geomA);
```

[illegible]

Note

[illegible]

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

[illegible]

Synopsis

```
geometry PostGIS_DropBBox(geometry geomA);
```


ST Intersects

Note



Note!

[illegible]

 This method supports Circular Strings and Curves.



```
--This example drops bounding boxes where the cached box is not correct
--The force to ST_AsBinary before applying Box2D forces a ↵
    recalculation of the box, and Box2D applied to the table ↵
    geometry always
-- returns the cached bounding box.
    UPDATE sometable
SET geom = PostGIS_DropBBOX(geom)
WHERE Not (Box2D(ST_AsBinary(geom)) = Box2D(geom));

    UPDATE sometable
SET geom = PostGIS_AddBBOX(geom)
WHERE Not PostGIS HasBBOX(geom);
```



PostGIS AddBBox, PostGIS HasBBox, Box2D

7.23.3 PostGIS HasBBox

PostGIS HasBBox — `boolean`, `boolean`.

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 的几何引擎，它封装了 CGAL 和 C++ (wrapper) 库。它提供了许多函数，用于处理几何数据。

SFCGAL 的官方网站是 <http://www.sfcgal.org>。您可以在该网站上找到有关 SFCGAL 的更多信息。SFCGAL 是 PostGIS 的一个扩展，您可以通过安装 PostGIS 来使用它。

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、Triangles 或 TIN，则将其转换为 Polyhedral 表面。

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 — 返回几何体的朝向 (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 — 计算几何体的面积

Synopsis

double precision **CG_Area**(geometry geom);

返回

计算几何体的面积。

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.

示例

```
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))');
```

返回

ST_3DArea, ST_Area

8.2.7 CG_3DArea

CG_3DArea — 3 维几何体的表面积。返回 0 或正数。

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 几何体的表面积。返回 0 或正数。返回 0 表示空几何体。

```
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 — Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.

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)

￼￼

A closed polyhedral surface has volume. **CG_MakeSolid** makes the solid representation explicit.

```
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
-----+-----
1 | 1
```

￼￼

CG_3DArea, **CG_MakeSolid**, **CG_IsSolid**

8.2.9 ST_ForceLHR

ST_ForceLHR — LHR(Left Hand Reverse; ￼￼￼) ￼￼￼￼￼￼.

Synopsis

geometry **ST_ForceLHR**(geometry geom);



Warning

ST_ForceLHR is deprecated as of 3.5.0. Use **CG_ForceLHR** instead.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



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

8.2.10 ST_IsPlanar

ST_IsPlanar — □□□□□□□□□□□□□□.

Synopsis

```
boolean ST_IsPlanar(geometry geom);
```



Warning

ST_IsPlanar is deprecated as of 3.5.0. Use **CG_IsPlanar** instead.

2.2.0 2.1.0 2.1



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



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

8.2.11 ST_IsSolid

ST IsSolid —                                                                    

Synopsis

```
boolean ST_IsSolid(geometry geom1);
```


☒☒



Warning

ST_IsSolid is deprecated as of 3.5.0. Use **CG_IsSolid** instead.

2.2.0 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 — Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.

Synopsis

```
float ST_Volume(geometry geom1);
```

￼



Warning

ST_Volume is deprecated as of 3.5.0. Use **CG_Volume** instead.

2.2.0 ￼￼￼￼￼￼￼.

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 9.1 (same as ST_3DVolume)

￼

A closed polyhedral surface has volume. **ST_MakeSolid** makes the solid representation explicit.

```
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
-----+-----
1 | 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)'));
```

```
POINT(2.5 2.5)
```

☒☒

[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);
```

f

```
SELECT CG_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
```

t

☒☒

[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)))');
```

t

￼￼

[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).

Usage

```
SELECT ST_AsText(CG_Difference('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'::geometry, 'LINESTRING ↵
(0 0, 2 2)'::geometry));
```

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

See also

[ST_3DDifference](#), [ST_Difference](#)

8.3.5 ST_3DDifference

ST_3DDifference — 3 **DEPRECATED**.

Synopsis

geometry **ST_3DDifference**(geometry geom1, geometry geom2);

See also

**Warning**

ST_3DDifference is deprecated as of 3.5.0. Use **CG_3DDifference** instead.

geom2 **DEPRECATED** geom1 **DEPRECATED**.

2.2.0 **DEPRECATED**.



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 维几何体相减。

Synopsis

geometry **CG_3DDifference**(geometry geom1, geometry geom2);

几何体

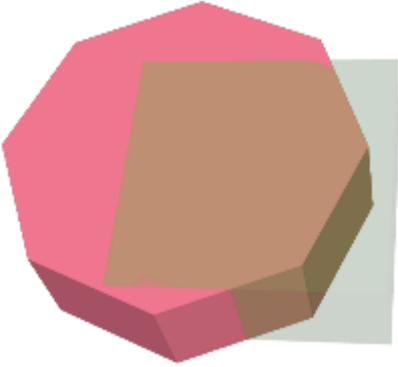
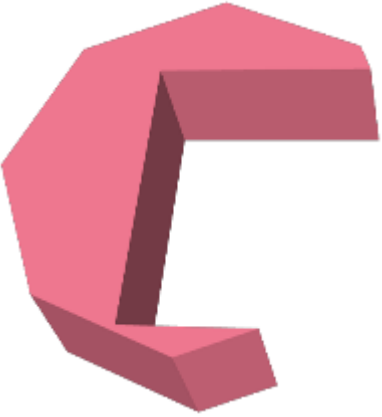
geom2 从 geom1 中减去。

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).

几何体

PostGIS **ST_AsX3D** 将 3 维几何体转换为 **X3Dom HTML** 格式。HTML 格式。HTML 格式。

<pre>SELECT CG_Extrude(ST_Buffer(↵ ST_GeomFromText('POINT(100 90)'), 50, 'quad_segs=2'), 0, 0, 30) AS geom1, ↵ CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'), 50, 'quad_segs=2'), POINT(80 80),↵ geom2;</pre>  3 维几何体。geom2 从 geom1 中减去。	<pre>SELECT CG_3DDifference(geom1,geom2)</pre>  geom2 从 geom1 中减去。	CG_3DDifference



[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).



```
SELECT CG_Distance('LINESTRING(0.0 0.0,-1.0 -1.0)', 'LINESTRING(3.0 4.0,4.0 5.0)');
```

```
5
```



[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);



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).



```
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))');
```

1



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 — 计算 3D 凸包。

Synopsis

```
geometry CG_3DConvexHull(geometry geom1);
```

返回

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).

示例

```
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;
```


2.1.0 新增功能。

- ✔ This method needs SFCGAL backend.
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

8.3.12 CG_3DIntersection

CG_3DIntersection — 3 维相交。

Synopsis

geometry **CG_3DIntersection**(geometry geom1, geometry geom2);

返回

geom1 与 geom2 的交集。

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=2'), 0, 0, 30) AS geom1,
    CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'), 50, 'quad_segs=1'), 0, 0, 30) AS geom2;↵
SELECT CG_3DIntersection(geom1,geom2)
CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'), 50, 'quad_segs=1'), 0, 0, 30) AS geom2 ) As t;↵
```

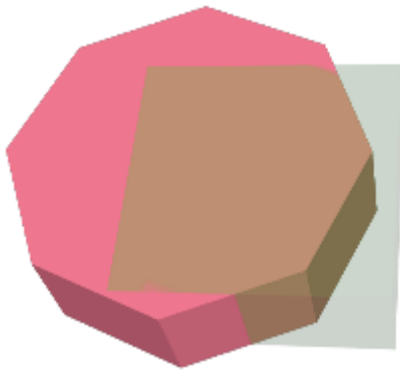
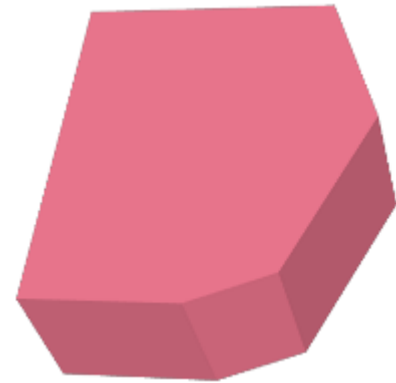


图 3 两个相交的多边形。 geom2 在 geom1 之上。



geom1 与 geom2 的交集。

3 维多边形交集

```
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;↵
```

```
LINESTRING Z (1 1 8,0.5 0.5 8)
```

图 4 (图 3 中的) 交集 Z

```
SELECT ST_AsText(
CG_3DIntersection(
ST_GeomFromText('POLYHEDRALSURFACE Z( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'),
'POLYGON Z ((0 0 0, 0 0 0.5, 0 0.5 0.5, 0 0.5 0, 0 0 0))'::geometry))
```

```
TIN Z (((0 0 0,0 0 0.5,0 0.5 0.5,0 0 0)),((0 0.5 0,0 0 0,0 0.5 0.5,0 0.5 0)))
```

图 5 图 3 中的交集 (ST_Dimension 返回 3 维)

```
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 ST_AsText(CG_Union('POINT(.5 0)', 'LINESTRING(-1 0,1 0)'));
```

```
LINESTRING(-1 0,0.5 0,1 0)
```

￼￼

ST_3DUnion, **ST_AsBinary**

8.3.14 ST_3DUnion

ST_3DUnion — Perform 3D union.

Synopsis

```
geometry ST_3DUnion(geometry geom1, geometry geom2);  
geometry ST_3DUnion(geometry set g1field);
```





Warning

ST_3DUnion is deprecated as of 3.5.0. Use **CG_3DUnion** instead.

2.2.0 .

Availability: 3.3.0 aggregate variant was added

-  This method needs SFCGAL backend.
-  This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
-  This function supports 3d and will not drop the z-index.
-  This function supports Polyhedral surfaces.
-  This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Aggregate variant: returns a geometry that is the 3D union of a rowset of geometries. The ST_3DUnion() function is an “aggregate” function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.

8.3.15 CG_3DUnion

CG_3DUnion — Perform 3D union using postgis_sfcgal.

Synopsis

```
geometry CG_3DUnion(geometry geom1, geometry geom2);  
geometry CG_3DUnion(geometry set g1field);
```



Availability: 3.5.0

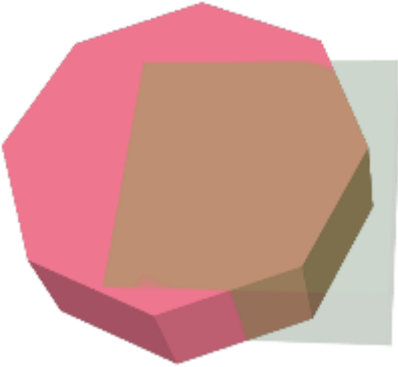
-  This method needs SFCGAL backend.
-  This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

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

Aggregate variant: returns a geometry that is the 3D union of a rowset of geometries. The `CG_3DUnion()` function is an “aggregate” function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the `SUM()` and `AVG()` functions do and like most aggregates, it also ignores NULL geometries.

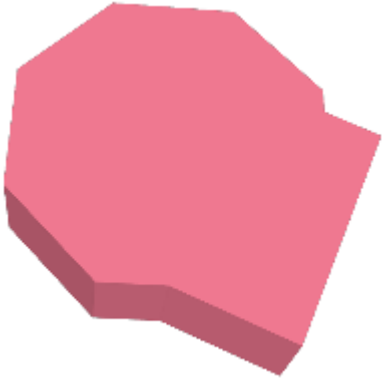
PostGIS **ST_AsX3D** 3 **X3Dom HTML** **HTML**.

```
SELECT CG_Extrude(ST_Buffer(
  ST_GeomFromText('POINT(100 90)'), 50, 'quad_segs=2'), 0, 0, 30) AS geom1,
CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'), 50, 'quad_segs=2'), 0, 0, 30) AS geom2;
```



3 `geom2`

```
SELECT CG_3DUnion(geom1,geom2)
AS t;
```



`geom1` `geom2`

CG_Extrude, ST_AsX3D, CG_3DIntersection CG_3DDifference

8.3.16 ST_AlphaShape

ST_AlphaShape — Computes an Alpha-shape enclosing a geometry

Synopsis

geometry **ST_AlphaShape**(geometry geom, float alpha, boolean allow_holes = false);

☒☒

**Warning**

ST_AlphaShape is deprecated as of 3.5.0. Use **CG_AlphaShape** instead.

Computes the **Alpha-Shape** of the points in a geometry. An alpha-shape is a (usually) concave polygonal geometry which contains all the vertices of the input, and whose vertices are a subset of the input vertices. An alpha-shape provides a closer fit to the shape of the input than the shape produced by the **convex hull**.

8.3.17 CG_AlphaShape

CG_AlphaShape — Computes an Alpha-shape enclosing a geometry

Synopsis

geometry **CG_AlphaShape**(geometry geom, float alpha, boolean allow_holes = false);

☒☒

Computes the **Alpha-Shape** of the points in a geometry. An alpha-shape is a (usually) concave polygonal geometry which contains all the vertices of the input, and whose vertices are a subset of the input vertices. An alpha-shape provides a closer fit to the shape of the input than the shape produced by the **convex hull**.

The “closeness of fit” is controlled by the alpha parameter, which can have values from 0 to infinity. Smaller alpha values produce more concave results. Alpha values greater than some data-dependent value produce the convex hull of the input.

Note

Following the CGAL implementation, the alpha value is the *square* of the radius of the disc used in the Alpha-Shape algorithm to “erode” the Delaunay Triangulation of the input points. See **CGAL Alpha-Shapes** for more information. This is different from the original definition of alpha-shapes, which defines alpha as the radius of the eroding disc.

The computed shape does not contain holes unless the optional `allow_holes` argument is specified as true.

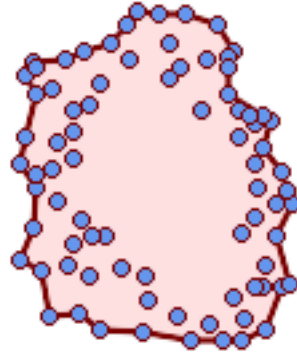
This function effectively computes a concave hull of a geometry in a similar way to **ST_ConcaveHull**, but uses CGAL and a different algorithm.

Availability: 3.5.0 - requires SFCGAL >= 1.4.1.



This method needs SFCGAL backend.

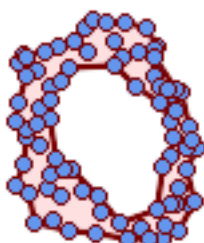
囧囧



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));
```

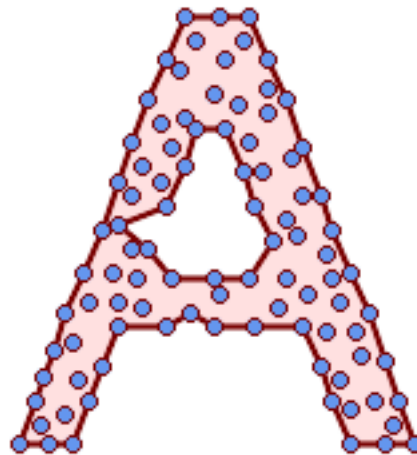
```
MULTIPOLYGON(((89 53,88 58,85 71,83 72,75 84,76 88,72 95,64 97,57 97,52 98,49 95,45 90,39 ←
  88,34 86,26 85,24 82,27 77,24 67,23 60,27 54,26 44,23 36,28 33,30 22,37 23,43 19,53 ←
  18,65 16,73 16,78 16,84 19,88 29,90 30,87 42,91 50,89 53)))
```



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))
```

```
MULTIPOLYGON(((89 53,88 58,85 71,83 72,75 84,76 88,72 95,64 97,57 97,52 98,49 95,45 90,39 ←
  88,34 86,26 85,24 82,27 77,24 67,23 60,27 54,26 44,23 36,28 33,30 22,43 19,53 18,65 ←
  16,73 16,78 16,84 19,90 30,87 42,91 50,89 53),(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 61,36 68)))
```



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));

MULTIPOLYGON(((168 36,163 51,157 69,150 84,143 100,139 113,132 131,126 149,119 164,112 180,101 180,88 180,81 164,74 149,68 133,63 118,57 100,50 84,43 69,39 55,35 45,32 36,26 20,37 20,46 20,52 36,57 49,63 64,81 64,90 69,99 64,114 64,132 64,139 49,143 36,150 20,163 20,174 20,168 36),(81 109,88 124,92 138,103 138,110 122,114 109,121 96,112 82,99 82,83 82,74 93,81 109)))
```

☒☒

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.

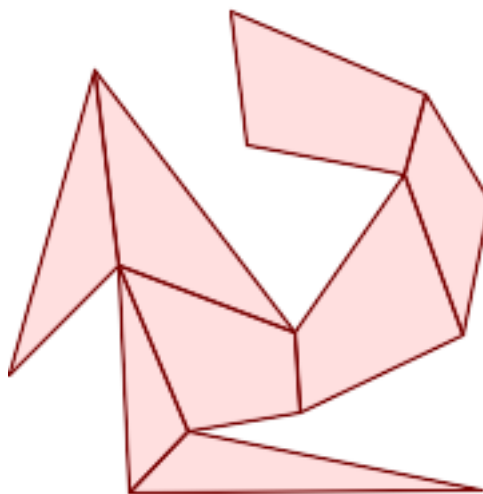
Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

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);
 geometry **CG_ApproximateMedialAxis**(geometry geom, boolean projected);

☒☒

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).

When projected is `true`, free endpoints of the medial axis are extended to reach the polygon boundary (projected medial axis). Requires SFCGAL 2.3.0+. When built against an older SFCGAL version, a notice is emitted and the non-projected result is returned instead.

Availability: 3.5.0

Availability: 3.7.0 - projected parameter. Requires SFCGAL $\geq$ 2.3.0 for projected result; falls back to non-projected with a notice on older versions.



Note

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



This method needs SFCGAL backend.



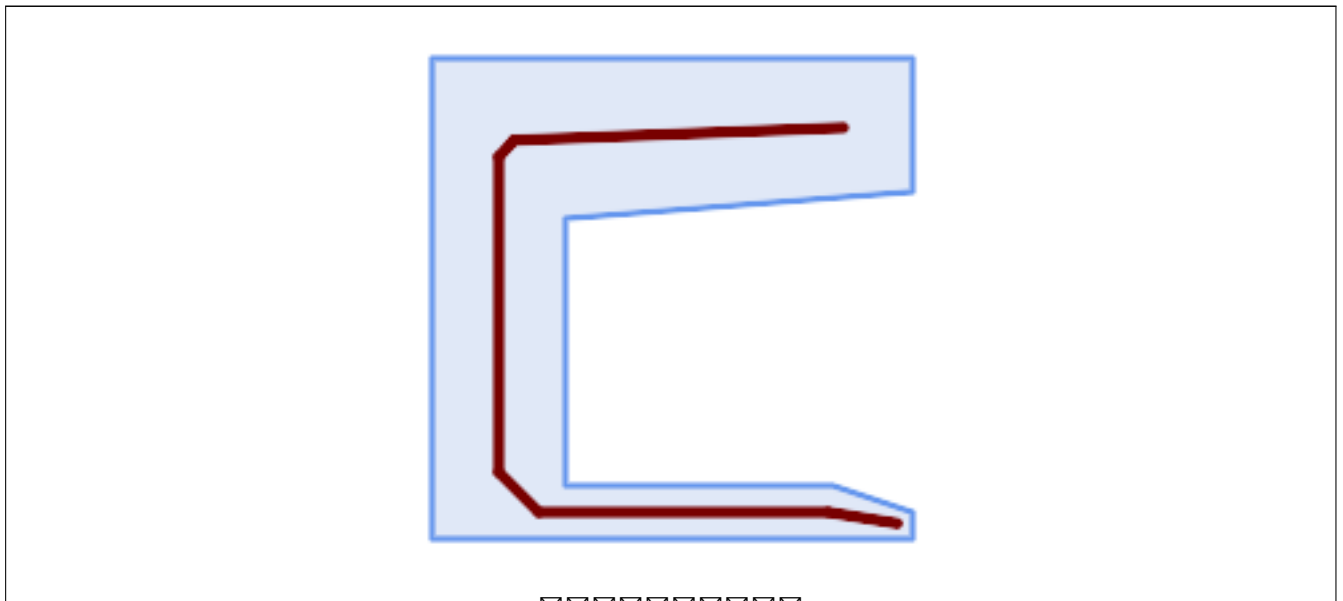
This function supports Polyhedral surfaces.



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

例

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



例

This example computes the projected medial axis with free endpoints extended to the polygon boundary.

```
SELECT ST_AsText(CG_ApproximateMedialAxis('POLYGON((0 0,2 0,2 1,0 1,0 0))', true));
```

```
MULTILINESTRING((0 0.5,0.5 0.5,1.5 0.5,2 0.5))
```

例

[CG_StraightSkeleton](#), [CG_StraightSkeletonPartition](#)

8.3.21 CG_GenerateFlatRoof

CG_GenerateFlatRoof — Generate a flat (box) roof from a footprint polygon.

Synopsis

geometry **CG_GenerateFlatRoof**(geometry geom, float8 height);

☒☒

Generate a flat (box-shaped) roof as a 3D PolyhedralSurface Z from a 2D footprint polygon. The roof height above the ground plane is given by height (default 3.0).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.



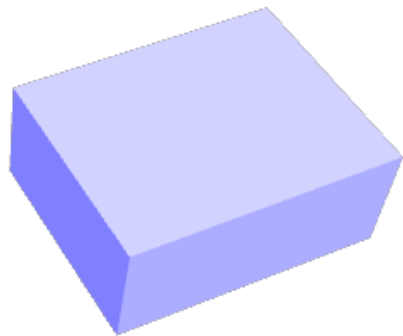
This method needs SFCGAL backend.



This function supports Polyhedral surfaces.

☒☒

```
SELECT ST_AsText(
  CG_GenerateFlatRoof(
    'POLYGON((0 0,5 0,5 4,0 4,0 0))',
    2.0));
-- POLYHEDRALSURFACE Z (...)
```



☒☒

CG_GenerateHippedRoof, **CG_GenerateGableRoof**, **CG_GenerateSkillionRoof**, **CG_GenerateRoof**

8.3.22 CG_GenerateHippedRoof

CG_GenerateHippedRoof — Generate a hipped roof from a footprint polygon.

Synopsis

geometry **CG_GenerateHippedRoof**(geometry geom, float8 height);

￼￼

Generate a hipped roof as a 3D PolyhedralSurface Z from a 2D footprint polygon. All sides slope upward to meet at a ridge or apex. The roof height is given by height (default 3.0).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.



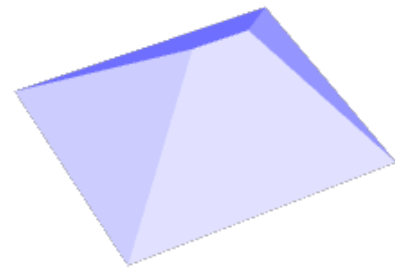
This method needs SFCGAL backend.



This function supports Polyhedral surfaces.

￼￼

```
SELECT ST_AsText(
  CG_GenerateHippedRoof(
    'POLYGON((0 0,5 0,5 4,0 4,0 0))',
    2.0));
-- POLYHEDRALSURFACE Z (...)
```



￼￼

[CG_GenerateFlatRoof](#), [CG_GenerateGableRoof](#), [CG_GenerateSkillionRoof](#), [CG_GenerateRoof](#)

8.3.23 CG_GenerateGableRoof

CG_GenerateGableRoof — Generate a gable roof from a footprint polygon.

Synopsis

geometry **CG_GenerateGableRoof**(geometry geom, float8 height, float8 slope_angle);

￼￼

Generate a gable roof as a 3D PolyhedralSurface Z from a 2D footprint polygon. Two opposite sides are vertical gable ends; the other two sides slope to a horizontal ridge. height sets the ridge height above the ground (default 3.0) and slope_angle controls the pitch in degrees (default 30.0). The SRID of the input geometry is preserved.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.



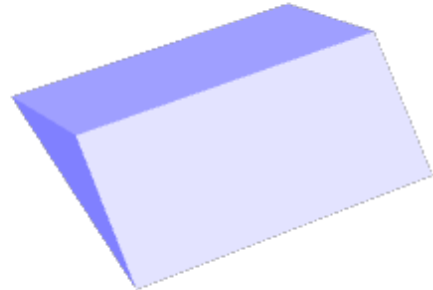
This method needs SFCGAL backend.



This function supports Polyhedral surfaces.

☒☒

```
SELECT ST_CoordDim(
  CG_GenerateGableRoof(
    'POLYGON((0 0,5 0,5 4,0 4,0 0))',
    2.0, 30.0));
-- 3
```



☒☒

[CG_GenerateFlatRoof](#), [CG_GenerateHippedRoof](#), [CG_GenerateSkillionRoof](#), [CG_GenerateRoof](#)

8.3.24 CG_GenerateSkillionRoof

`CG_GenerateSkillionRoof` — Generate a skillion (single-slope) roof from a footprint polygon.

Synopsis

geometry **CG_GenerateSkillionRoof**(geometry geom, float8 height, float8 slope_angle, integer primary_edge_index);

☒☒

Generate a skillion (single-slope, shed-style) roof as a 3D `PolyhedralSurface` Z from a 2D footprint polygon. The entire roof surface slopes in one direction. `height` sets the maximum ridge height (default 3.0), `slope_angle` sets the pitch in degrees (default 30.0), and `primary_edge_index` selects which polygon edge defines the high side (0-based, default 0).

When the combination of `height` and `slope_angle` would cause a side wall to vanish (the slope reaches the far edge before the full height is achieved), that face is omitted from the result. This means the number of output patches can vary depending on the parameters —a lower height relative to the footprint depth produces more faces than a taller, steeper configuration.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.



This method needs SFCGAL backend.



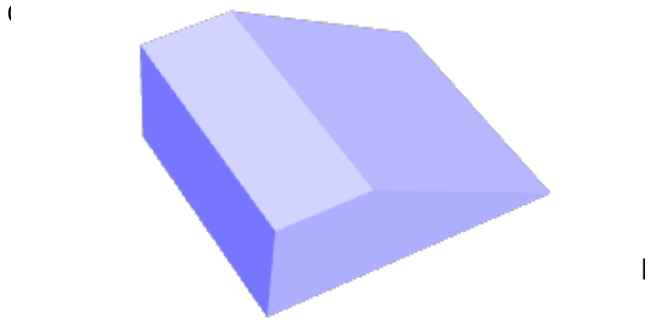
This function supports Polyhedral surfaces.

☒☒

Height 2.0 produces 6 faces (all walls present); height 4.0 produces 5 faces (one side wall is filtered out because the slope reaches the far edge before the full height).

This example uses height 2.0 and returns 6 patches.

```
SELECT ST_NumPatches(  
  'POLYGON((0 0,5 0,5 4,0 4,0 0))',  
  2.0));  
-- 6  
  
-- height 4.0 -> 5 patches (one face ←  
  filtered)  
SELECT ST_NumPatches(  
  CG_GenerateSkillionRoof(  
    'POLYGON((0 0,5 0,5 4,0 4,0 0))',  
    4.0));  
-- 5
```



☒☒

[CG_GenerateFlatRoof](#), [CG_GenerateHippedRoof](#), [CG_GenerateGableRoof](#), [CG_GenerateRoof](#)

8.3.25 CG_GenerateRoof

CG_GenerateRoof — Generate a roof of the requested type from a footprint polygon.

Synopsis

geometry **CG_GenerateRoof**(geometry geom, text roof_type, float8 height, float8 slope_angle, integer primary_edge_index);

☒☒

Generate a roof as a 3D PolyhedralSurface Z from a 2D footprint polygon. The roof_type parameter selects the roof style (default 'HIPPED'):

- FLAT —flat box roof.
- HIPPED —hipped roof (all sides slope to apex).
- GABLE —gable roof (two sloping sides, two vertical ends).
- SKILLION —single-slope (shed) roof.

`height` sets the roof height (default 3.0), `slope_angle` sets the pitch in degrees (default 30.0), and `primary_edge_index` selects the reference edge for skillion roofs (default 0).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.



This method needs SFCGAL backend.



This function supports Polyhedral surfaces.



```
SELECT ST_AsText(CG_GenerateRoof('POLYGON((0 0,4 0,4 4,0 4,0 0))', 'HIPPED', 2.0, 30.0, 0)) ←
;
-- POLYHEDRALSURFACE Z (((0 4 0,4 4 0,4 0 0,0 0 0,0 4 0)), ...)

SELECT ST_AsText(CG_GenerateRoof('POLYGON((0 0,4 0,4 4,0 4,0 0))', 'FLAT', 2.0));
-- POLYHEDRALSURFACE Z (((0 0 0,0 4 0,4 4 0,4 0 0,0 0 0)), ...)
```



[CG_GenerateFlatRoof](#), [CG_GenerateHippedRoof](#), [CG_GenerateGableRoof](#), [CG_GenerateSkillionRoof](#)

8.3.26 CG_PolygonRepair

`CG_PolygonRepair` — Repair an invalid polygon or multipolygon.

Synopsis

geometry **CG_PolygonRepair**(geometry geom, text rule);



Repair an invalid polygon or multipolygon using CGAL's 2D Polygon Repair algorithm. Returns a `MultiPolygon`.

The rule parameter controls the repair strategy:

- `EVEN_ODD` (default) —areas covered an odd number of times are kept. Available with CGAL 6.0+.
- `NON_ZERO` —areas with non-zero winding number are kept. Requires CGAL 6.1+.
- `UNION` —union of all input polygons. Requires CGAL 6.1+.
- `INTERSECTION` —intersection of all input polygons. Requires CGAL 6.1+.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0 and CGAL >= 6.0.



This method needs SFCGAL backend.



This function supports Polyhedral surfaces.

❏

This example repairs a bowtie self-intersecting polygon.

```
SELECT ST_AsText(CG_PolygonRepair('POLYGON((0 0,2 2,2 0,0 2,0 0))'));
-- MULTIPOLYGON(((0 0,1 1,0 2,0 0)),((1 1,2 0,2 2,1 1)))

-- Keep all winding area with UNION rule (requires CGAL 6.1 or later)
SELECT ST_AsText(CG_PolygonRepair('POLYGON((0 0,2 2,2 0,0 2,0 0))', 'UNION'));
-- MULTIPOLYGON(((0 0,1 1,0 2,0 0)))
```

❏

[ST_IsValid](#), [ST_MakeValid](#)

8.3.27 ST_ConstrainedDelaunayTriangles

ST_ConstrainedDelaunayTriangles — Return a constrained Delaunay triangulation around the given input geometry.

Synopsis

geometry **ST_ConstrainedDelaunayTriangles**(geometry g1);

❏



Warning

[ST_ConstrainedDelaunayTriangles](#) is deprecated as of 3.5.0. Use [CG_ConstrainedDelaunayTriangles](#) instead.

Return a [Constrained Delaunay triangulation](#) around the vertices of the input geometry. Output is a TIN.



This method needs SFCGAL backend.

2.1.0 简体中文.



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

8.3.28 CG_ConstrainedDelaunayTriangles

CG_ConstrainedDelaunayTriangles — Return a constrained Delaunay triangulation around the given input geometry.

Synopsis

geometry **CG_ConstrainedDelaunayTriangles**(geometry g1);

☒☒

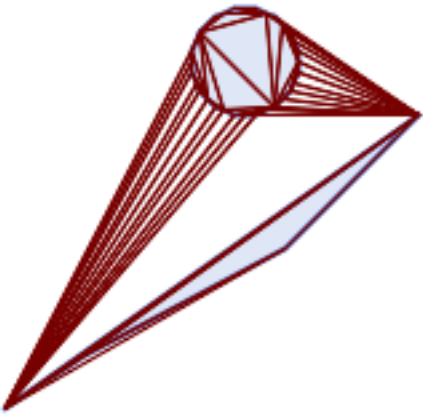
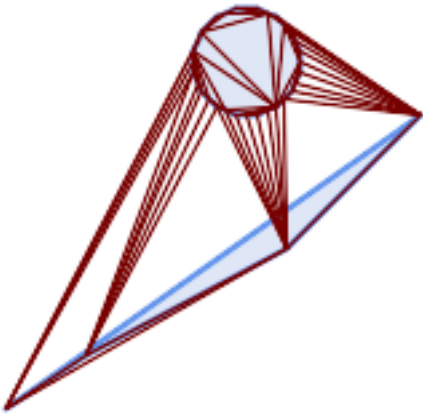
Return a **Constrained Delaunay triangulation** around the vertices of the input geometry. Output is a TIN.

✔ 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_Tessellate, ST_ConcaveHull, ST_Dump

8.3.29 ST_Extrude

ST_Extrude — ☒☒☒☒☒☒☒☒☒☒.

Synopsis

geometry **ST_Extrude**(geometry geom, float x, float y, float z);



Warning
ST_Extrude is deprecated as of 3.5.0. Use **CG_Extrude** instead.

2.1.0

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

8.3.30 CG_Extrude

CG_Extrude —

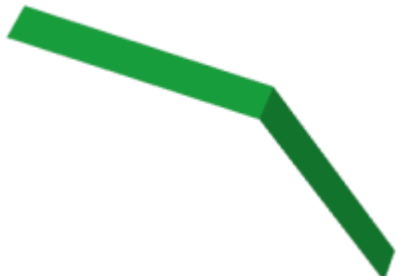
Synopsis

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).

PostGIS **ST_AsX3D** 3 **X3Dom HTML** **HTML**

<pre>SELECT ST_Buffer(quad_segs=2'),0,0,30);</pre> 	<pre>SELECT ST_GeomFromText('POINT(100 90)'), CG_Extrude(0, 0, 30)</pre> 
<pre>SELECT ST_GeomFromText('LINESTRING(50 50, 100 90, 95 150)')</pre> 	<pre>SELECT CG_Extrude(ST_GeomFromText(' LINESTRING(50 50, 100 90, 95 150)'),0,0,10));</pre> 



ST AsX3D, CG ExtrudeStraightSkeleton

8.3.31 CG_ExtrudeStraightSkeleton

CG ExtrudeStraightSkeleton — Straight Skeleton Extrusion

Synopsis

geometry **CG_ExtrudeStraightSkeleton**(geometry geom, float roof_height, float body_height = 0);

⊠

Computes an extrusion with a maximal height of the polygon geometry.

Note



Perhaps the first (historically) use-case of straight skeletons: given a polygonal roof, the straight skeleton directly gives the layout of each tent. If each skeleton edge is lifted from the plane a height equal to its offset distance, the resulting roof is "correct" in that water will always fall down to the contour edges (the roof's border), regardless of where it falls on the roof. The function computes this extrusion aka "roof" on a polygon. If the argument `body_height > 0`, so the polygon is extruded like with `CG_Extrude(polygon, 0, 0, body_height)`. The result is an union of these polyhedralsurfaces.

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.

⊠

```
SELECT ST_AsText(CG_ExtrudeStraightSkeleton('POLYGON (( 0 0, 5 0, 5 5, 4 5, 4 4, 0 4, 0 0 ) ←
, (1 1, 1 2, 2 2, 2 1, 1 1))', 3.0, 2.0));
```

```
POLYHEDRALSURFACE Z (((0 0 0,0 4 0,4 4 0,4 5 0,5 5 0,5 0 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 ←
0,1 1 0)),((0 0 0,0 0 2,0 4 2,0 4 0,0 0 0)),((0 4 0,0 4 2,4 4 2,4 4 0,0 4 0)),((4 4 0,4 ←
4 2,4 5 2,4 5 0,4 4 0)),((4 5 0,4 5 2,5 5 2,5 5 0,4 5 0)),((5 5 0,5 5 2,5 0 2,5 0 0,5 5 ←
0)),((5 0 0,5 0 2,0 0 2,0 0 0,5 0 0)),((1 1 0,1 1 2,2 1 2,2 1 0,1 1 0)),((2 1 0,2 1 2,2 ←
2 2,2 0,2 1 0)),((2 2 0,2 2 2,1 2 2,1 2 0,2 2 0)),((1 2 0,1 2 2,1 1 2,1 1 0,1 2 0)) ←
,((0 4 2,0 0 2,0.5 0.5 2.5,0.5 2.5 2.5,1 3 3,0 4 2)),((0 0 2,5 0 2,3.5 1.5 3.5,2.5 0.5 ←
2.5,0.5 0.5 2.5,0 0 2)),((5 0 2,5 5 2,4.5 4.5 2.5,4.5 3.5 2.5,3.5 2.5 3.5,3.5 1.5 3.5,5 ←
0 2)),((5 5 2,4 5 2,4.5 4.5 2.5,5 5 2)),((4 5 2,4 4 2,4.5 3.5 2.5,4.5 4.5 2.5,4 5 2)) ←
,((4 4 2,0 4 2,1 3 3,3 3 3,3.5 2.5 3.5,4.5 3.5 2.5,4 4 2)),((2 1 2,1 1 2,0.5 0.5 2.5,2.5 ←
0.5 2.5,2 1 2)),((1 1 2,1 2 2,0.5 2.5 2.5,0.5 0.5 2.5,1 1 2)),((1 2 2,2 2 2,3 3 3,1 3 ←
3,0.5 2.5 2.5,1 2 2)),((2 2 2,2 1 2,2.5 0.5 2.5,3.5 1.5 3.5,3.5 2.5 3.5,3 3 3,2 2 2)))
```

⊠

ST_Extrude, **CG_ExtrudeStraightSkeleton**, **CG_StraightSkeleton**, **CG_StraightSkeletonPartition**

8.3.32 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.

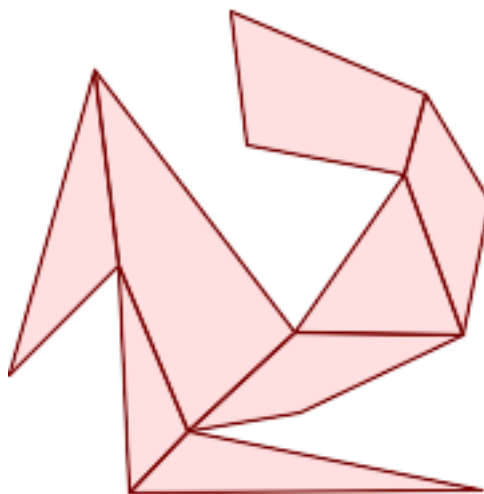
Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

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)))
```


返回一个 2 维多边形 (x, y, z) 的 Minkowski 和。 3 维多边形, z 坐标 0 的 2 维多边形 Minkowski 和。 2 维多边形 Minkowski 和。

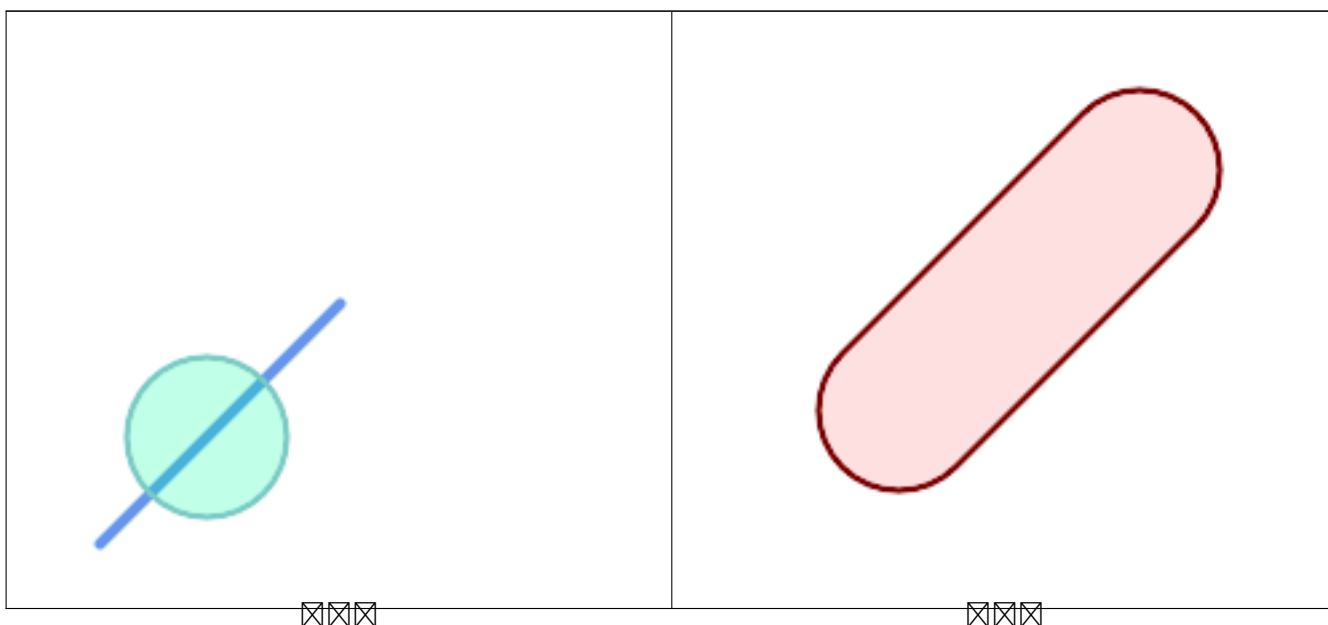
使用 **CGAL 2D MinkowskiSum** 函数。

Availability: 3.5.0

✔ This method needs SFCGAL backend.

图例

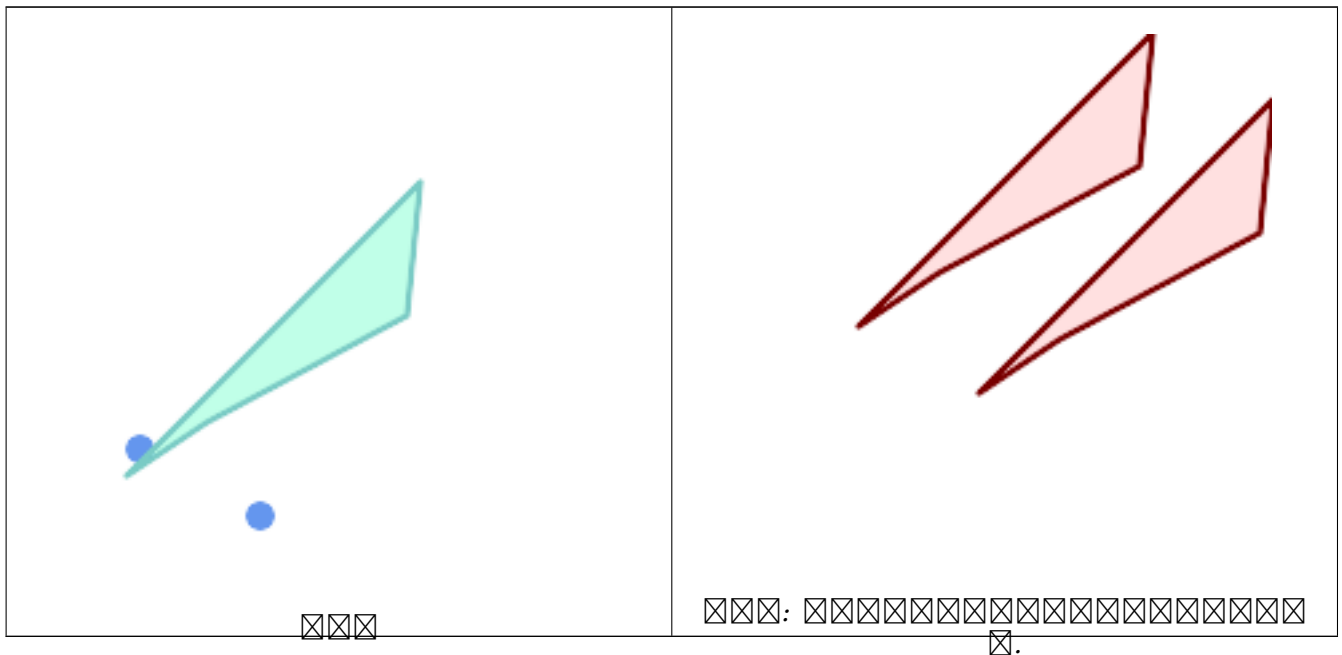
图例: 2D MinkowskiSum 的结果



```
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)))
```

图例: 点, 多边形, 结果



```
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))
)
```

8.3.35 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);

图例



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.

8.3.36 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

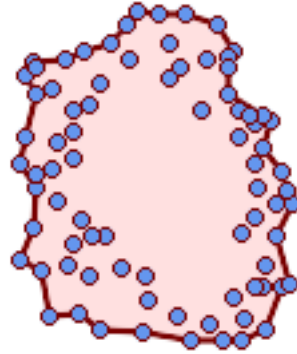
The result will not contain holes unless the optional `allow_holes` argument is specified as true.

Availability: 3.5.0 - requires SFCGAL $\geq$ 1.4.1.



This method needs SFCGAL backend.

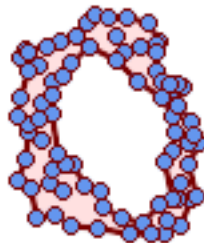
☒☒



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));

MULTIPOLYGON(((89 53,88 58,83 64,85 71,83 72,75 77,75 84,76 88,72 95,64 97,57 97,52 98,49 ←
  95,45 90,39 88,34 86,26 85,24 82,27 77,24 67,23 60,27 54,26 44,23 36,28 33,30 22,37 ←
  23,43 19,53 18,65 16,73 16,78 16,84 19,88 29,90 30,87 42,91 50,89 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));

MULTIPOLYGON(((89 53,88 58,83 64,85 71,83 72,75 77,75 84,76 88,72 95,64 97,57 97,52 98,49 ←
  95,45 90,39 88,34 86,26 85,24 82,27 77,24 67,23 60,27 54,26 44,23 36,28 33,30 22,37 ←
  23,43 19,53 18,65 16,73 16,78 16,84 19,88 29,90 30,87 42,91 50,89 53),(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 61,36 68)))
```

☒☒

[ST_ConcaveHull](#), [CG_AlphaShape](#)

8.3.37 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.

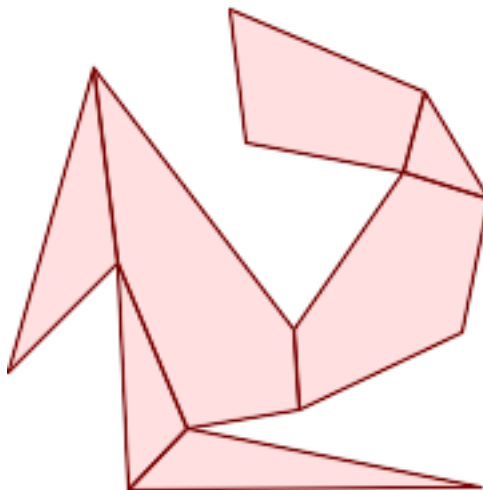
Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

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](#)

8.3.38 CG_StraightSkeleton

`CG_StraightSkeleton` — ￼￼￼￼￼￼￼ (straight skeleton) ￼￼￼￼￼.

Synopsis

geometry **CG_StraightSkeleton**(geometry geom, boolean use_distance_as_m = false);



Availability: 3.5.0

Requires SFCGAL >= 1.3.8 for option use_distance_as_m

-  This method needs SFCGAL backend.
-  This function supports 3d and will not drop the z-index.
-  This function supports Polyhedral surfaces.
-  This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



```
SELECT CG_StraightSkeleton(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, 190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```

```
SELECT ST_AsText(CG_StraightSkeleton('POLYGON((0 0,1 0,1 1,0 1,0 0))', true));
```

```
MULTILINESTRING M ((0 0 0,0.5 0.5 0.5),(1 0 0,0.5 0.5 0.5),(1 1 0,0.5 0.5 0.5),(0 1 0,0.5 0.5 0.5));
```

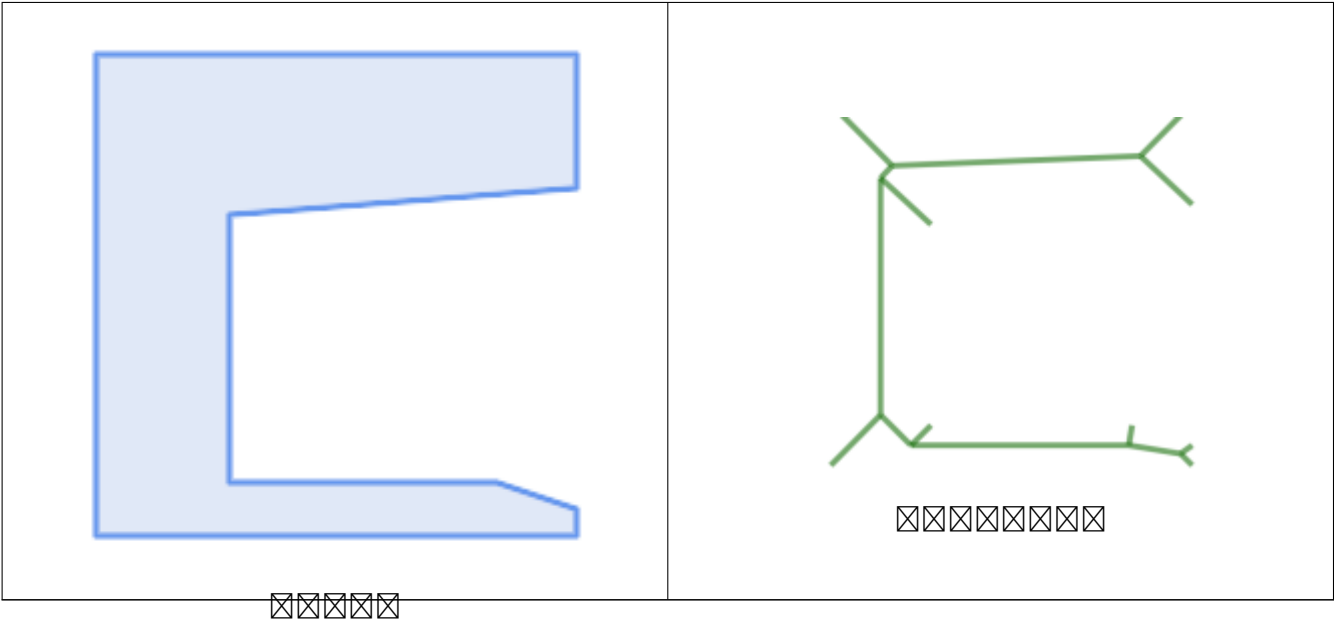
Note that valid inputs with rings that touch at a single point will raise an error.

```
SELECT CG_StraightSkeleton('POLYGON((0 0, 3 0, 3 3, 0 3, 0 0), (0 0, 1 2, 2 1, 0 0))');
```

NOTICE: During straight_skeleton(A) :

NOTICE: with A: POLYGON((0/1 0/1,3/1 0/1,3/1 3/1,0/1 3/1,0/1 0/1),(0/1 0/1,1/1 2/1,2/1 1/1,0/1 0/1))

ERROR: straight skeleton of Polygon with point touching rings is not implemented.



☒

[CG_StraightSkeletonPartition](#), [CG_ExtrudeStraightSkeleton](#)

8.3.39 ST_StraightSkeleton

ST_StraightSkeleton — 计算 (straight skeleton) 骨架。

Synopsis

geometry **ST_StraightSkeleton**(geometry geom);

☒



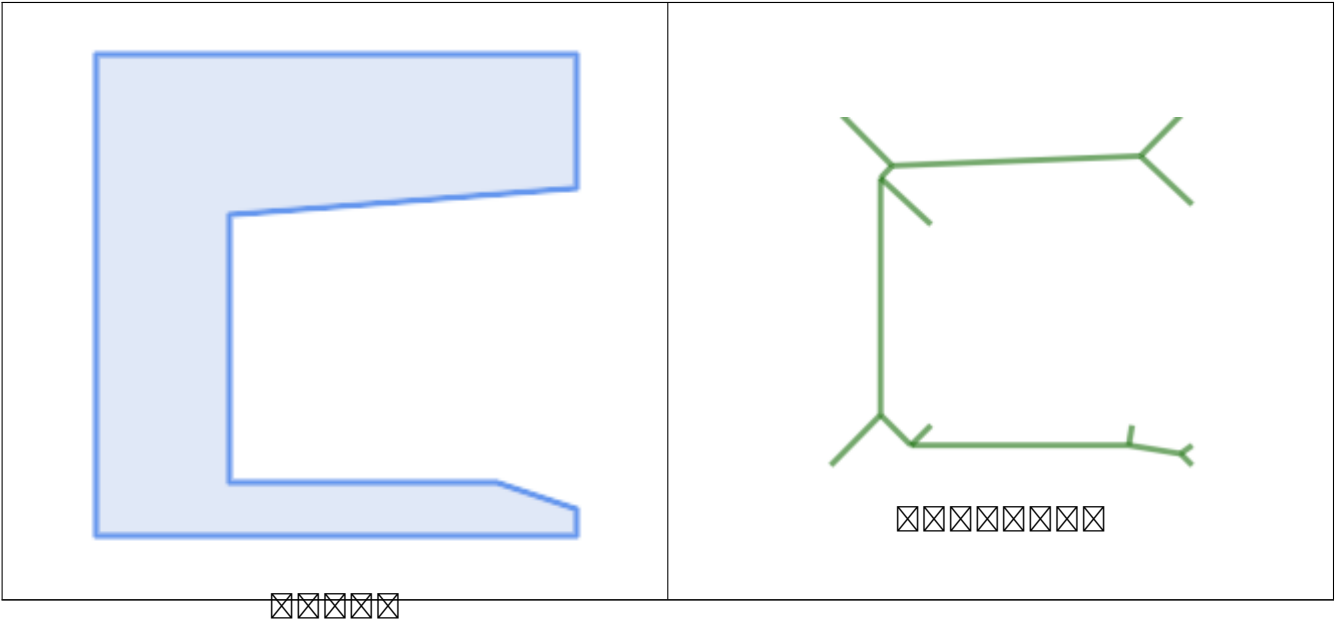
Warning
[ST_StraightSkeleton](#) is deprecated as of 3.5.0. Use [CG_StraightSkeleton](#) instead.

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).

图例

```
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

8.3.40 ST_Tessellate

ST_Tessellate — 将多边形分解为三角形 (tessellation) 返回 TIN 或 TIN 集合。
图例

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。
图例



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).

8.3.41 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
```

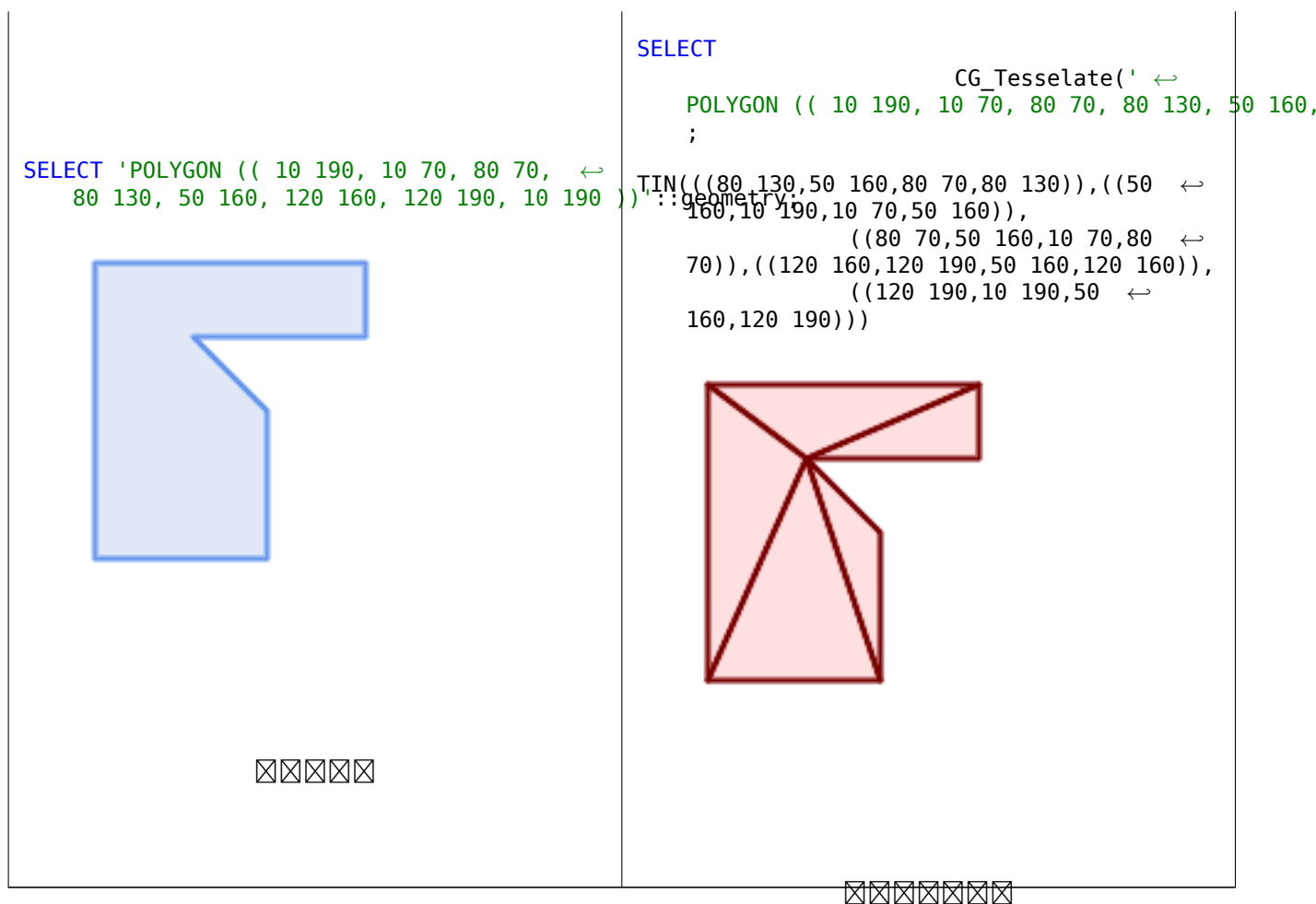


繁體中文

```
SELECT CG_Tessellate(ST_GeomFromText('
POLYHEDRALSURFACE Z( ((0 0 0, 0 0 1, 0 1 1, 0 1
      ((0 0 0,
0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1
      ((1 1 0,
1 1 1, 1 0 1, 1 0 0, 1 1 0)),
      ((0 1 0,
0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1
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 1,0 0 1,0 0 1,0 0 0)),
0,0 0 0)),
      ((1 0 0,0 0 0,1 1
0,1 0 0)),((0 0 1,1 0 0,1 0 1,0 0 1)),
      ((0 0 1,0 0 0,1 0
0,0 0 1)),
      ((1 1 0,1 1 1,1 0
1,1 1 0)),((1 0 0,1 1 0,1 0 1,1 0 0)),
      ((0 1 0,0 1 1,1 1
1,0 1 0)),((1 1 0,0 1 0,1 1 1,1 1 0)),
      ((0 1 1,1 0 1,1 1
1,0 1 1)),((0 1 1,0 0 1,1 0 1,0 1 1)))
```



繁體中文



CG_ConstrainedDelaunayTriangles, ST_DelaunayTriangles, ST_TriangulatePolygon

8.3.42 CG_Triangulate

CG Triangulate — Triangulates a polygonal geometry

Synopsis

```
geometry CG_Triangulate( geometry geom );
```


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 ST_AsText(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))'));
```

st_astext

```
-----
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 0.2,0.8 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](#)

8.3.43 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.



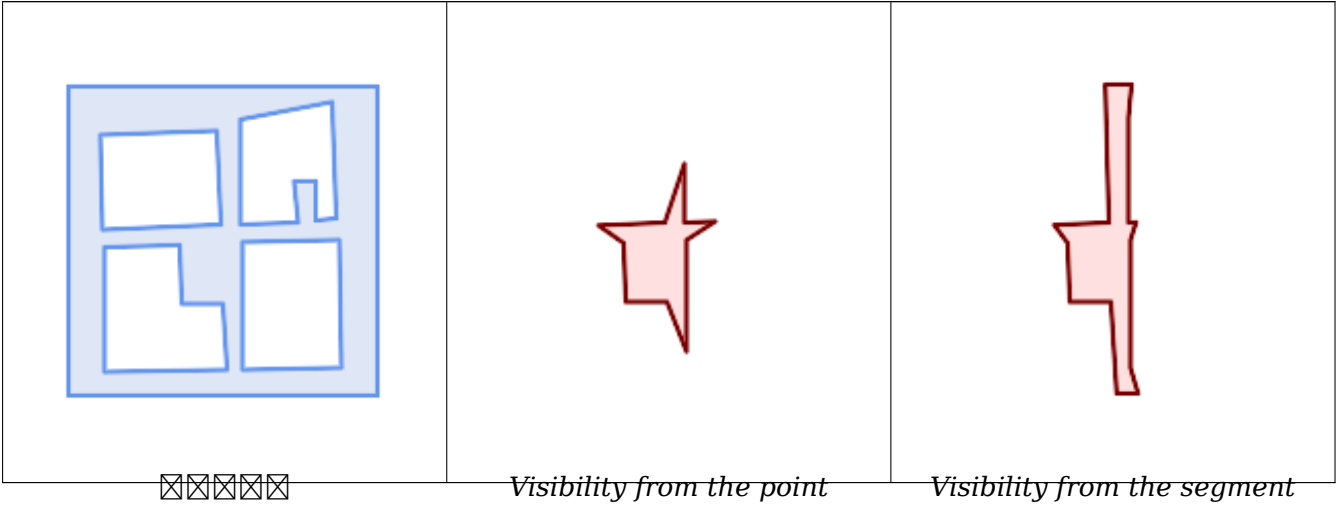
This function supports Polyhedral surfaces.



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);
```



8.3.44 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.

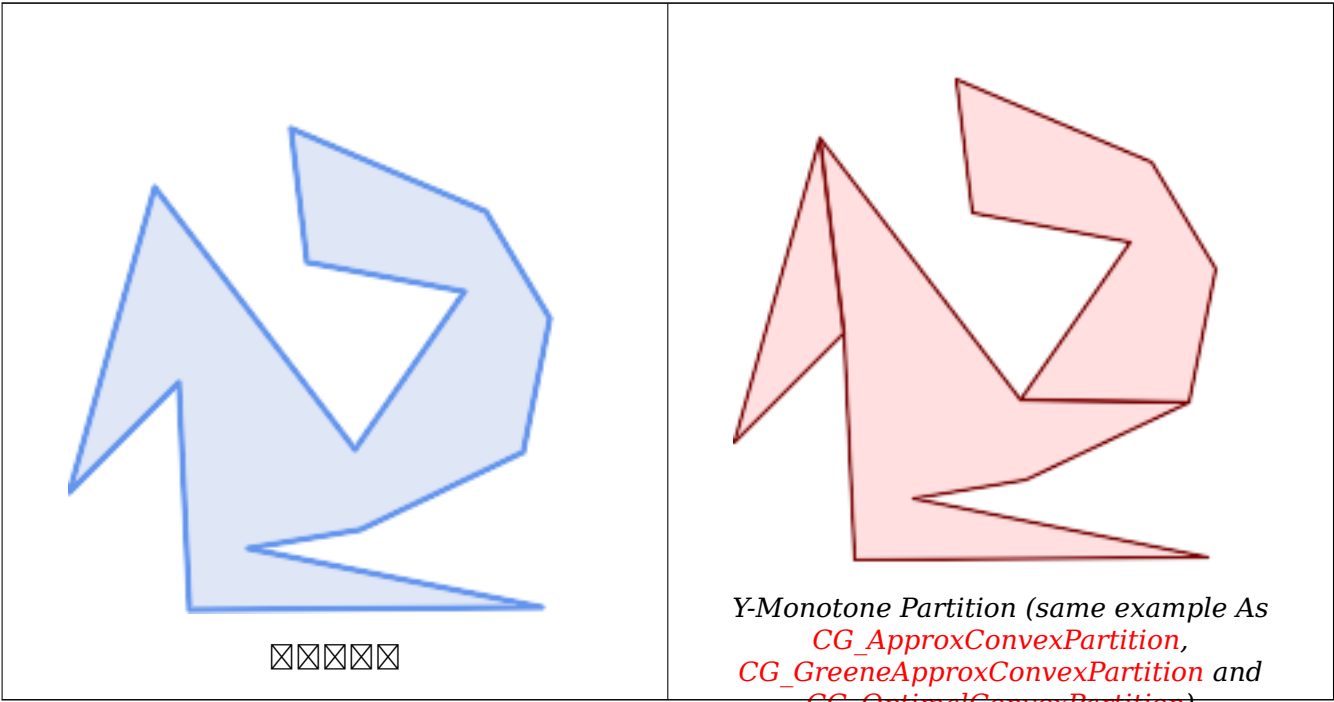


Note
A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. A y-monotone polygon is a polygon whose vertices $v_1, \dots, v_n$ can be divided into two chains $v_1, \dots, v_k$ 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*

8.3.45 CG_StraightSkeletonPartition

`CG_StraightSkeletonPartition` — Computes the straight skeleton partition of a polygon.

Synopsis

geometry **CG_StraightSkeletonPartition**(geometry geom, boolean auto_orientation);

Computes the straight skeleton partition of the input polygon geometry *geom*. The straight skeleton is a partitioning of the polygon into faces formed by tracing the collapse of its edges. If *auto_orientation* is set to true, the function will automatically adjust the orientation of the input polygon to ensure correct results.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0.

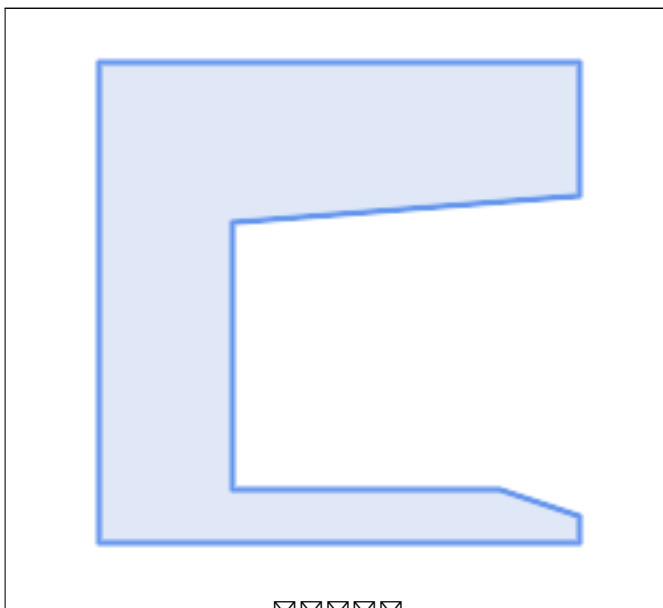
This method needs SFCGAL backend.

❏

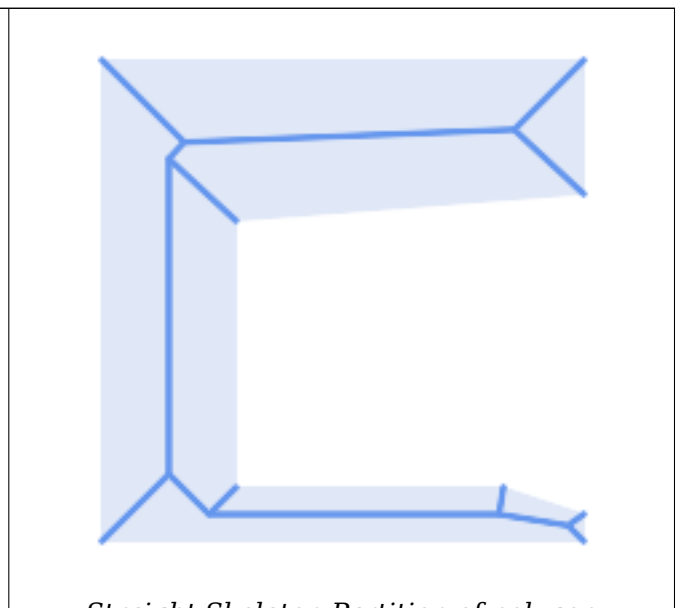
```
SELECT ST_AsText(CG_StraightSkeletonPartition('POLYGON((0 0, 4 0, 2 2, 0 0))', true));
```

```
POLYHEDRALSURFACE(((0 0,2 0.82842712474619,2 2)),((4 0,2 0.82842712474619,0 0)),((2 2,2 0.82842712474619,4 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 );
```



❏



Straight Skeleton Partition of polygon

❏

CG_StraightSkeleton, ST_Polygonize

8.3.46 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.

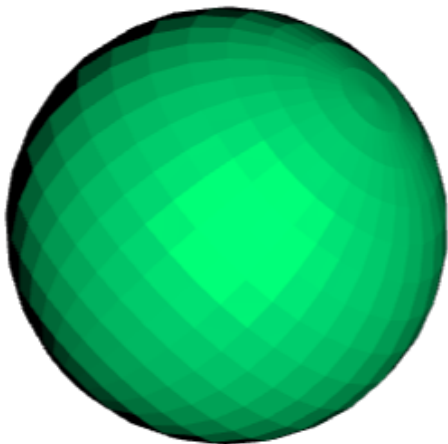
例

```
SELECT ST_AsText(CG_3DBuffer('POINT(0 0 0)', 1, 8, 0));
```

```
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
>', '');
```



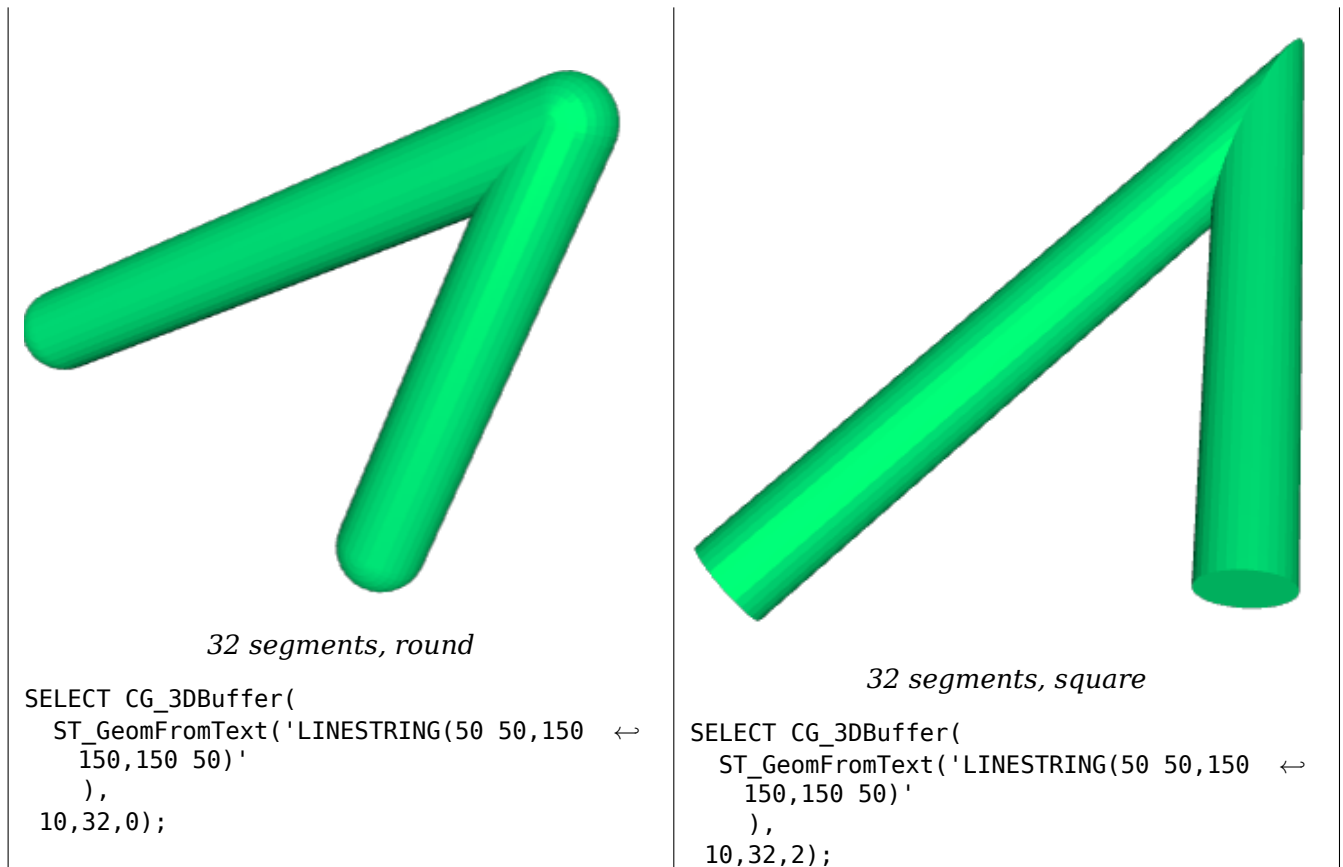
segments=32 (rounded buffer)

```
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](#)

8.3.47 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));
```

```
LINESTRING(6.123233995736766e-17 1, -1 6.123233995736766e-17)
```

❏

[CG_2DRotate](#), [ST_Rotate](#)

8.3.48 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));
```

```
POINT(2 1)
```

❏

[CG_Rotate](#), [CG_3DRotate](#)

8.3.49 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));
```

```
POINT Z (6.123233995736766e-17 1 0)
```

￼￼

CG_RotateX, **CG_RotateY**, **CG_RotateZ**

8.3.50 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));
```

```
POINT Z (0 6.123233995736766e-17 1)
```

￼￼

CG_RotateY, **CG_RotateZ**, **CG_3DRotate**

8.3.51 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));
```

```
POINT Z (6.123233995736766e-17 0 -1)
```



CG_RotateX, **CG_RotateZ**, **CG_3DRotate**

8.3.52 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));
```

```
POINT Z (6.123233995736766e-17 1 0)
```

❏

[CG_RotateX](#), [CG_RotateY](#), [CG_3DRotate](#)

8.3.53 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));
```

```
LINESTRING(2 2,4 4)
```

❏

[CG_3DScale](#), [CG_3DScaleAroundCenter](#), [ST_Scale](#)

8.3.54 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));
```

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



[CG_Scale](#), [CG_3DScaleAroundCenter](#)

8.3.55 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));
```

```
POINT Z (1.5 1.5 1.5)
```



[CG_Scale](#), [CG_3DScale](#)

8.3.56 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));
```

```
LINESTRING(2 0,3 1)
```



CG_3DTranslate, **ST_Translate**

8.3.57 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));
```

```
POINT Z (2 0 3)
```

❏❏

[CG_Translate](#), [ST_Translate](#)

8.3.58 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.

☒☒

This example simplifies 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));
```

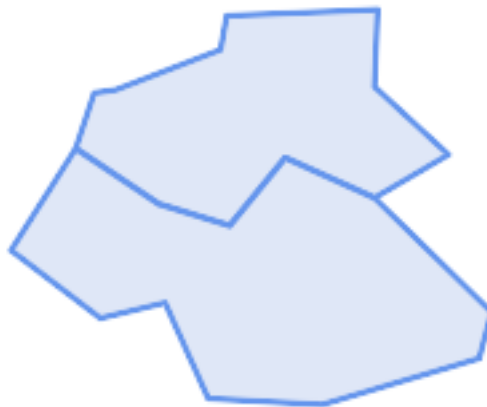
Simplify two geometries together, preserving Z and M values without preserving topology.

```
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 geometries together, preserving Z and M values and topology.

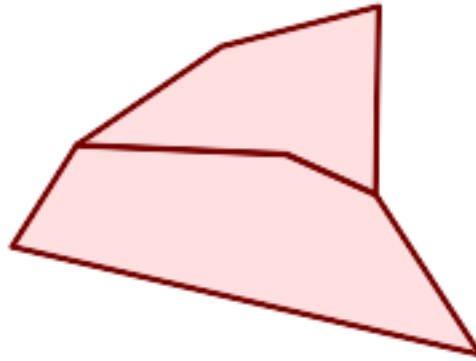
```
SELECT ST_AsText(CG_Simplify('GEOMETRYCOLLECTION ZM(LINESTRING ZM(-1 -1 3 4, 0 0 10 100, 1 1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150), POLYGON ZM((0 0 10 100, 1 1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150, 0 0 10 100)))', 2, true));
```

```
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;
```



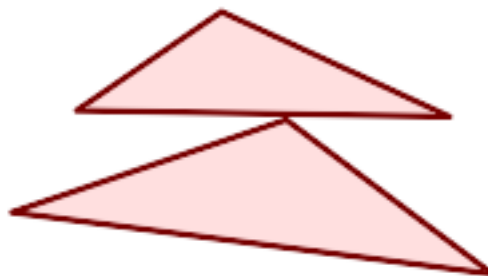
Originals geometries

```
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](#)

8.3.59 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.

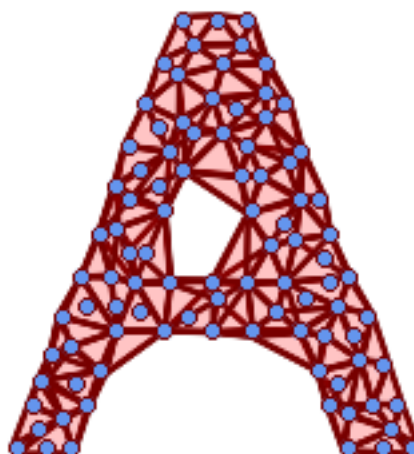
☒☒

```
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);
```

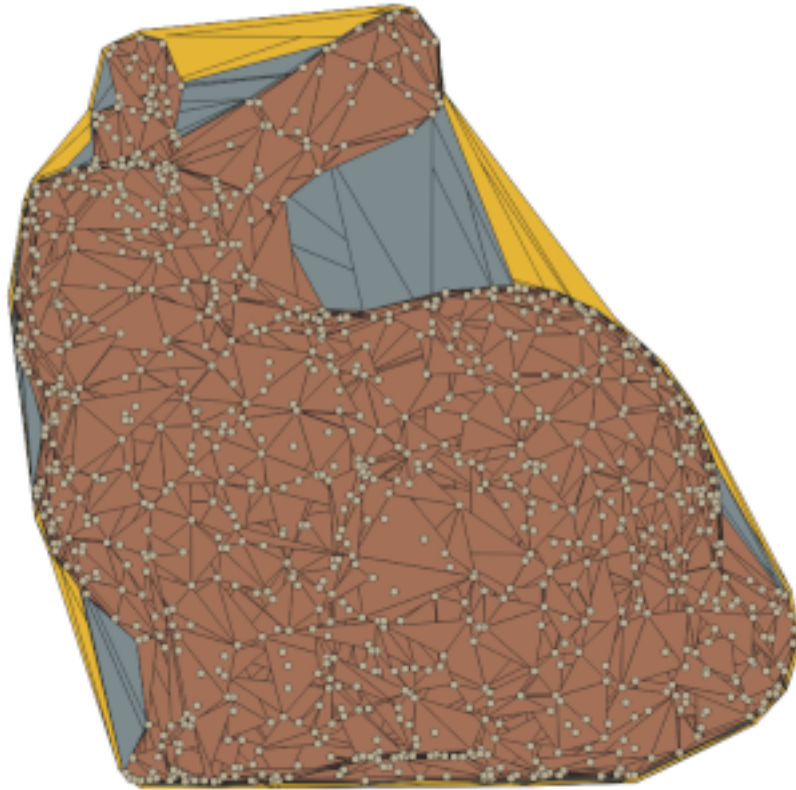



Alpha wrapping of a MultiPoint (same example As [CG_OptimalAlphaShape](#))

```
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

8.3.60 CG_NurbsCurveInterpolate

`CG_NurbsCurveInterpolate` — Creates an interpolating NURBS curve passing through all given data points

Synopsis

geometry **CG_NurbsCurveInterpolate**(geometry data_points, integer degree);



Creates a NURBS curve that interpolates through all provided data points. The resulting curve passes exactly through each point in the input geometry.

Parameters:

- `data_points` - A `LINestring` geometry containing the points to interpolate through.
- `degree` - Polynomial degree of the NURBS curve (typically 2 or 3).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.



This method needs SFCGAL backend.



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



This example creates a smooth curve interpolating through five points.

```
SELECT CG_NurbsCurveInterpolate(
  'LINESTRING(0 0, 1 2, 3 3, 5 1, 6 0)::geometry,
  3
);

-- 3D interpolation
SELECT CG_NurbsCurveInterpolate(
  'LINESTRING Z(0 0 0, 1 2 1, 3 3 0, 5 1 -1, 6 0 0)::geometry,
  2
);
```



CG_NurbsCurveApproximate, **ST_MakeNurbsCurve**

8.3.61 CG_NurbsCurveApproximate

CG_NurbsCurveApproximate — Creates an approximating NURBS curve fitting data points within a tolerance

Synopsis

geometry **CG_NurbsCurveApproximate**(geometry data_points, integer degree, float8 tolerance, integer max_control_points=100);



Creates a NURBS curve that approximates the given data points within the specified tolerance. Unlike interpolation, the curve does not necessarily pass through all points but provides a smooth fit with fewer control points.

Parameters:

- **data_points** - A LINESTRING geometry containing the points to approximate.
- **degree** - Polynomial degree of the NURBS curve.
- **tolerance** - Maximum allowed distance between the curve and data points.
- **max_control_points** - Optional maximum number of control points (default: 100). Limits curve complexity.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.



This method needs SFCGAL backend.



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



This example approximates noisy data using a tolerance.

```
SELECT CG_NurbsCurveApproximate(  
    'LINESTRING(0 0, 1 2.1, 2 1.9, 3 3.2, 4 2.8, 5 1)'::geometry,  
    2,  
    0.5  
);  
  
-- Control maximum complexity  
SELECT CG_NurbsCurveApproximate(  
    'LINESTRING(0 0, 1 2, 2 2, 3 3, 4 3, 5 1, 6 0, 7 -1, 8 0)'::geometry,  
    3,  
    0.2,  
    20 -- max 20 control points  
);
```



CG_NurbsCurveInterpolate, ST_MakeNurbsCurve

8.3.62 CG_NurbsCurveDerivative

CG_NurbsCurveDerivative — Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter

Synopsis

geometry **CG_NurbsCurveDerivative**(geometry nurbscurve, float8 parameter, integer derivative_order);



Computes the derivative of a NURBS curve at a specific parameter value. The derivative represents the rate of change at that point:

- Order 1: Tangent vector (direction and speed)
- Order 2: Curvature vector (how the tangent changes)
- Order 3: Third-order derivative (supported maximum)

Parameters:

- nurbscurve - A NURBS curve geometry.
- parameter - Parameter value for evaluation (typically 0.0 to 1.0).
- derivative_order - Order of derivative to compute (1, 2, or 3).

Returns a POINT geometry representing the derivative vector at the specified parameter.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.



This method needs SFCGAL backend.



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

☒☒

This example returns the tangent vector at the curve midpoint.

```
WITH nurbs AS (  
    SELECT ST_MakeNurbsCurve(2, 'LINESTRING(0 0, 5 10, 10 0)::geometry') AS geom  
)  
SELECT CG_NurbsCurveDerivative(geom, 0.5, 1) FROM nurbs;  
  
-- Get curvature at start of curve  
WITH nurbs AS (  
    SELECT ST_MakeNurbsCurve(3, 'LINESTRING(0 0, 2 5, 5 5, 8 2, 10 0)::geometry') AS geom  
)  
SELECT CG_NurbsCurveDerivative(geom, 0.0, 2) FROM nurbs;  
  
-- Analyze tangent at multiple points  
WITH nurbs AS (  
    SELECT ST_MakeNurbsCurve(2, 'LINESTRING(0 0, 5 10, 10 0)::geometry') AS geom  
)  
SELECT  
    t.param,  
    CG_NurbsCurveDerivative(n.geom, t.param, 1) AS tangent  
FROM nurbs n  
CROSS JOIN (VALUES (0.0), (0.25), (0.5), (0.75), (1.0)) AS t(param);
```

☒☒

ST_Evaluate, ST_MakeNurbsCurve

Chapter 9

Topology

PostGIS represents a face, an edge, a node, and a topology.

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](#).

An example of a topologically based GIS database is the [US Census Topologically Integrated Geographic Encoding and Referencing System \(TIGER\)](#) database. If you want to experiment with PostGIS topology and need some data.

The PostGIS topology module has existed for a long time but was not always part of the official documentation. Extensive cleanup removed deprecated functions, fixed known usability issues, documented the features and functions, added new functionality, and improved SQL-MM compliance.

Visit the [PostGIS Topology Wiki](#) for more information.

The topology module is part of the PostGIS database.

SQL/MM Part 2 defines the ST_ functions, PostGIS implements the ST_ functions.

Topology support is built by default and can be disabled by specifying the `--without-topology` configure option at build time as described in [Chapter 2](#).

9.1 Topology Primitive Tables

The core primitives of any topology are stored in the `edge_data`, `node`, and `face` tables that live in the schema created by [CreateTopology](#). Each row of `edge_data` represents an oriented edge: it records a directed curve from `start_node` to `end_node` together with the identifier of the face encountered on the left of that direction (`left_face`) and the face encountered on the right (`right_face`). The same geometric segment may therefore appear twice—once for each orientation—when it belongs to two faces.

The `next_left_edge` and `next_right_edge` columns complete this orientation information by encoding how to keep walking around a face. They store signed integers whose absolute value is the identifier of the next oriented edge and whose sign determines whether the stored orientation has to be followed as-is or reversed when traversing. Formally, the following rules hold for every edge `e`:

- `abs(next_left_edge)` is the identifier of the edge reached by continuing around the face that lies to the left of `e`. If the value is positive the walk continues from the end node of `e` along the stored orientation of the referenced edge; if it is negative the referenced edge must be followed backwards so that the shared face remains on the walker's left.

- `abs(next_right_edge)` analogously follows the boundary of the face located on the right of `e`. A positive value means that the next edge is taken with its recorded orientation starting from the end node of `e`, whereas a negative value instructs to traverse the referenced edge in reverse, starting from its end node, so that the right-hand face is preserved.
- A zero value indicates that the edge is dangling on the corresponding side (for example an isolated edge whose incident face is the universal face 0). The `abs_next_left_edge` and `abs_next_right_edge` columns exposed by the edge view are convenience projections of these absolute values.

This representation is a variant of a doubly connected edge list and is exploited by many topology routines. Functions such as `GetRingEdges` and `ValidateTopology` rely on it to reconstruct face boundaries and to diagnose inconsistencies—hence the “invalid next_left_edge” and “invalid next_right_edge” diagnostics reported during validation. Constructors like `AddEdge` initialise the `next_*` attributes with trivial self references, while editing routines including `ST_AddEdgeModFace` and `ST_RemEdgeModFace` update the links as edges are inserted or removed. Other bulk operations (for example `Polygonize`) may intentionally leave the fields unset, which is why the documentation flags their behaviour explicitly.

9.2 国国国国国国

9.2.1 getfaceedges_returntype

`getfaceedges_returntype` — A composite type that consists of a sequence number and an edge number.

国国

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 国国国国国国`: 国国国国国国 SRID 国国国国国国 topology.topology 国国国国国国国国国国国国国国.
2. `edge 国国国国国国`: 国国国国国国国国国国.

9.2.2 TopoGeometry

`TopoGeometry` — A composite type representing a topologically defined geometry.

国国

国国国国国国国国国国, 国国国国国国 ID 国国国国国国国国国国国国国国国国国国. `TopoGeometry` 国国国国 国国国国 topology_id, layer_id, id, type 国国国国国国国国.

1. `topology_id 国国国国国国`: 国国国国国国 SRID 国国国国国国 topology.topology 国国国国国国国国国国国国国国.
2. `layer_id 国国国国国国`: `TopoGeometry` 国国国国国国国国国国国国国国 layer_id 国国. topology_id 国 layer_id 国国国国国国 topology.layers 国国国国国国国国国国 (unique reference) 国国国国国国.
3. `id 国国国国国国`: 国国国国国国国国国国国国国国国国国国国国国国, 国国国国国国国国国国国国.
4. 1 国国 4 国国 type 国国国国国国国国国国国国国国. 1: [国国] 国国, 2: [国国] 国国, 3: [国国] 国国国, 4: 国国国国.

数据类型

PostGIS 3.7.0alpha1 简体中文版

数据类型	数据类型
数据类型	数据类型

数据类型

CreateTopoGeom

9.2.3 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`.

数据类型

PostGIS 3.7.0alpha1 简体中文版. **ValidateTopology** 返回数据类型 `validate_topology_returntype`. ID 数据类型 `id1` 和 `id2` 数据类型 `validate_topology_returntype`.

- `error` 数据类型 (varchar) 数据类型: 数据类型.
 - 数据类型 (descriptor) 数据类型: `coincident nodes(数据类型)`, `edge crosses node(数据类型)`, `edge not simple(数据类型)`, `edge end node geometry mismatch(数据类型)`, `edge start node geometry mismatch(数据类型)`, `face overlaps face(数据类型)`, `face within face(数据类型)`.
- `id1` 数据类型: 数据类型 (edge)/数据类型 (face)/数据类型 (node) 数据类型.
- `id2` 数据类型: 数据类型 2 数据类型, 数据类型.

数据类型

ValidateTopology

9.3 数据类型

9.3.1 TopoElement

`TopoElement` — 数据类型 `TopoGeometry` 数据类型 2 数据类型.

数据类型

PostGIS 3.7.0alpha1 简体中文版 **TopoGeometry** 数据类型 1 数据类型 2 数据类型.

数据类型 `TopoGeometry` 数据类型, 数据类型 (topological primitive) 数据类型 (1: node, 2: edge, 3: face) 数据类型. 数据类型 `TopoGeometry` 数据类型 数据类型 `TopoGeometry` 数据类型.

**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}
```

This example shows what happens when casting a three-element array to topology.topoelement. The domain requires a two-element array, so the dimension check fails.

```
SELECT ARRAY[1,2,3]::topology.topoelement;
```

```
ERROR:  value for domain topology.topoelement violates check constraint "dimensions"
```

❏

[GetTopoGeomElements](#), [TopoElementArray](#), [TopoGeometry](#), [TopoGeom_addElement](#), [TopoGeom_remElement](#)

9.3.2 TopoElementArray

TopoElementArray — An array of TopoElement objects.

❏

1 TopoElement 对象, TopoElement 对象。

❏

```
SELECT '{{1,2},{4,3}}'::topology.topoelementarray AS tea;
```

```
{{1,2},{4,3}}
```

This is the more verbose equivalent.

```
SELECT ARRAY[ARRAY[1, 2], ARRAY[4, 3]]::topology.topoelementarray AS tea;
```

$\{\{1, 2\}, \{4, 3\}\}$

This example uses the array aggregate function packaged with topology.

```
SELECT topology.TopoElementArray_Agg(ARRAY[e, t] ORDER BY e, t) As tea
FROM generate_series(1, 4) As e CROSS JOIN generate_series(1, 3) As t;
```

$$\{\{1,1\},\{1,2\},\{1,3\},\{2,1\},\{2,2\},\{2,3\},\{3,1\},\{3,2\},\{3,3\},\{4,1\},\{4,2\},\{4,3\}\}$$

```
SELECT '{{1,2,4},{3,4,5}}'::topology.topoelementarray AS tea;
```

ERROR: value for domain topology.topoelementarray violates check constraint "dimensions"



TopoElement, GetTopoGeomElementArray, TopoElementArray_Agg

9.4 TopoGeometry

9.4.1 AddTopoGeometryColumn

AddTopoGeometryColumn — `TopoGeometry` `topology.layer` `layer id`.

Synopsis

```
integer AddTopoGeometryColumn(name topology_name, name schema_name, name table_name,
name column_name, varchar feature_type, integer child_layer);
integer AddTopoGeometryColumn(name topology_name, regclass tab, name column_name, integer
layer id, varchar feature type, integer child layer);
```



1. `TopoGeometry` 数据类型。 `TopoGeometry` 数据类型用于存储拓扑几何数据。

 2. `AddTopoGeometryColumn()` 函数。 `AddTopoGeometryColumn()` 函数用于添加拓扑几何列。

topology.layer

```
[child_layer] 00000000 (00 NULL 0000) 00, 00000000 (0000000000000000) 0
0 TopoGeometry 00000000. 00000000, 000000 (child_layer 0000 TopoGeometry 000
0000) 00 TopoGeometry 00000000.
```

1. `TopoGeometryColumn` (AddTopoGeometryColumn `table_name` ID `column_name`) `table_name` TopoGeometry `column_name`.

Valid feature_types are: POINT, MULTIPOINT, LINE, MULTILINE, POLYGON, MULTIPOLYGON, COLLECTION

Availability: 1.1

❏❏

This example creates the new table in the `ma_topo` schema. The table could be created 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');
```

❏❏

[DropTopoGeometryColumn](#), [toTopoGeom](#), [CreateTopology](#), [CreateTopoGeom](#)

9.4.2 RenameTopoGeometryColumn

`RenameTopoGeometryColumn` — Renames a topogeometry column

Synopsis

`topology.layer` **`RenameTopoGeometryColumn`**(regclass `layer_table`, name `feature_column`, name `new_name`)

❏❏

This function changes the name of an existing TopoGeometry column ensuring metadata information about it is updated accordingly.

Availability: 3.4.0

❏❏

```
SELECT topology.RenameTopoGeometryColumn('public.parcels', 'topogeom', 'tgeom');
```

❏❏

[AddTopoGeometryColumn](#), [RenameTopology](#)

9.4.3 DropTopology

`DropTopology` — 删除拓扑: `topology.topology` 名称, 删除 geometry_columns 表中的记录。

Synopsis

integer **`DropTopology`**(varchar `topology_schema_name`);

❏

❏ topology.topology ❏, ❏ geometry_columns ❏
❏. ❏. ❏.

Availability: 1.1

❏

ma_topo ❏ topology.topology ❏ geometry_columns ❏
❏.

```
SELECT topology.DropTopology('ma_topo');
```

❏

DropTopoGeometryColumn

9.4.4 RenameTopology

RenameTopology — Renames a topology

Synopsis

varchar **RenameTopology**(varchar old_name, varchar new_name);

❏

Renames a topology schema, updating its metadata record in the topology.topology table.

Availability: 3.4.0

❏

Rename a topology from topo_stage to topo_prod.

```
SELECT topology.RenameTopology('topo_stage', 'topo_prod');
```

❏

CopyTopology, RenameTopoGeometryColumn

9.4.5 DropTopoGeometryColumn

DropTopoGeometryColumn — schema_name ❏ table_name ❏ Topogeometry ❏
❏ topology.layer ❏.

Synopsis

text **DropTopoGeometryColumn**(varchar schema_name, varchar table_name, varchar column_name);

参数

schema_name 表名 table_name 拓扑几何 Topogeometry 拓扑 topology.layer 拓扑
 拓扑拓扑拓扑拓扑. 拓扑拓扑拓扑拓扑. 表: 拓扑拓扑拓扑拓扑拓扑拓扑
 拓扑拓扑拓扑 NULL 拓扑拓扑.

Availability: 1.1

示例

```
SELECT topology.DropTopoGeometryColumn('ma_topo', 'parcel_topo', 'topo');
```

参见

[AddTopoGeometryColumn](#)

9.4.6 FixCorruptTopoGeometryColumn

FixCorruptTopoGeometryColumn — Fixes topogeometry corruption caused by upgrade to postgis_topology 3.6.0 and higher

Synopsis

text **FixCorruptTopoGeometryColumn**(name layerSchema, name layerTable, name layerColumn);

参数

When upgrading from PostGIS topology <3.6.0 to version >3.6.0+, the topogeometry column definition was changed. This caused corruption in topogeometries created before the upgrade. This function fixes this corruption in affected tables.

Availability: 3.6.1

示例

Fix all topology columns

```
SELECT topology.FixCorruptTopoGeometryColumn(schema_name, table_name, feature_column)
FROM topology.layer;
```

参见

[UpgradeTopology](#)

9.4.7 Populate_Topology_Layer

Populate_Topology_Layer — Adds missing entries to topology.layer table by reading metadata from topo tables.

Synopsis

setof record **Populate_Topology_Layer**();

Adds missing entries to the topology.layer table by inspecting topology constraints on tables. This function is useful for fixing up entries in topology catalog after restores of schemas with topo data.

It returns the list of entries created. Returned columns are schema_name, table_name, feature_column.

2.3.0

```
SELECT CreateTopology('strk_topo');
CREATE SCHEMA strk;
CREATE TABLE strk.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('strk_topo', 'strk', 'parcels', 'topo', 'POLYGON');
```

This query returns no records because the feature is already registered.

```
SELECT *
FROM topology.Populate_Topology_Layer();
```

Rebuild the topology layer catalog.

```
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

9.4.8 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');
```

```
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
```

9.4.9 ValidateTopology

ValidateTopology — Returns a set of `validatetopology_returntype` objects detailing issues with topology.

Synopsis

setof `validatetopology_returntype` **ValidateTopology**(varchar toponame, geometry bbox);

⌘⌘

Returns a set of `validatetopology_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 .
invalid edge(❌❌❌❌❌❌❌❌)	Identifier of the edge.		An edge geometry is invalid. See ST_IsValid .
edge not simple(❌❌❌❌❌❌❌❌)	Identifier of the edge.		An edge geometry has self-intersections. See ST_IsSimple .
edge crosses edge(❌❌❌❌❌❌❌❌❌❌❌❌❌)	Identifier of first edge.	Identifier of second edge.	Two edges have an interior intersection. See ST_Relate .
edge start node geometry mismatch(❌❌❌❌❌❌❌❌❌❌❌❌❌)	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(❌❌❌❌❌❌❌❌❌❌❌❌❌)	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(❌❌❌❌❌❌❌❌)	Identifier of the orphaned face.		No edge reports an existing face on either of its sides (left_face, right_face).
face has no rings(❌❌❌❌❌❌❌❌)	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.

	id1	id2	Meaning
isolated node has containing_face	Identifier of the ill-defined node.		A node which is not reported as being on the boundary of any edges is lacking the indication of a containing face.
isolated node has wrong containing_face	Identifier of the misrepresented node.		A node which is not reported as being on the boundary of any edges indicates a containing face which is not the actual face containing it. See GetFaceContaining-Point .
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 ☒☒☒☒☒☒☒☒☒☒☒.

[illegible]

2.2.0 'edge crosses node' id1 id2

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	

`validate_topology_returntype`

9.4.10 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`

9.4.11 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;
```

```
MULTIPOINT(9 6,9 14)
```

❏

MakeTopologyPrecise

9.4.12 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.

Using a `gridSize` larger than the smallest edge extent raises an exception rather than collapsing the edge into an empty geometry. This safeguards adjacent topological relationships by preventing precision snapping from removing primitives altogether.



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
);
```

￼￼

ValidateTopologyPrecision, **ValidateTopology**

9.4.13 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_parcel', 'feature'));
```

```
city_data
```

￼￼

FindLayer

9.4.14 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'));
```

```
1
```

￼￼

FindTopology

9.4.15 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');
```

```
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
```

9.4.16 UpgradeTopology

UpgradeTopology — Upgrades the specified topology to support large ids (int8) for topology and primitive ids.

Synopsis

void **UpgradeTopology**(name toponame);

￼￼

Takes a topology name and upgrades it to support large ids (int8) for topology and primitive ids. The function upgrades the following: - face (face_id column from int4 to int8, face_id_seq from int4 to int8) - node (node_id column from int4 to int8, containing_face column from int4 to int8, node_id_seq from int4 to int8) - edge_data (edge_id column from int4 to int8, edge_data_edge_id_seq from int4 to int8, left_face and right_face columns from int4 to int8, start_node and end_node columns from int4 to int8, next_left_edge and next_right_edge columns from int4 to int8) - relation (topogeo_id column from int4 to int8, element_id from int4 to int8) - topology (useslargeids column set to true)

Availability: 3.6.0

￼￼

```
SELECT topology.UpgradeTopology('city_data');
```

9.5 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.

9.6 ￼￼￼￼￼

9.6.1 CreateTopology

CreateTopology — Creates a new topology schema and registers it in the topology.topology table.

Synopsis

integer **CreateTopology**(name topology_schema_name, integer srid, double precision prec=0, boolean hasz=false, integer topoid=0, boolean useslargeids=false);

国国

Creates a new topology schema with name `topology_name` and registers it in the `topology.topology` table. Topologies must be uniquely named. The topology tables (`edge_data`, `face`, `node`, and `relation`) are created in the schema. It returns the id of the topology.

The `srid` is the **spatial reference system** SRID for the topology. The SRID defaults to -1 (unknown) if not specified.

The tolerance `prec` is measured in the units of the spatial reference system. The tolerance defaults to 0.

`hasz` defaults to false if not specified.

`topoid` optional explicit identifier (allows deterministic topology id assignment, needs to be unique)

`useslargeids` optional, defaults to false. If true, the topology will be created to support large ids (int8) for topology and primitive ids.

This is similar to the SQL/MM **ST_InitTopoGeo** but has more functionality.

Availability: 1.1

Enhanced: 2.0 added support for `hasZ`.

Enhanced: 3.6.0 added `topoid` and `usesLargeIDs` parameters.

国国

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;
```

2

国国

Section 4.5, **ST_InitTopoGeo**

9.6.2 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](#)

9.6.3 ST_InitTopoGeo

`ST_InitTopoGeo` — Creates a new topology schema and registers it in the `topology.topology` table.

Synopsis

text **ST_InitTopoGeo**(varchar topology_schema_name);

❏

This is the SQL-MM equivalent of [CreateTopology](#). It lacks options for spatial reference system and tolerance. it returns a text description of the topology creation, instead of the topology id.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.17

❏

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation;
```

```
Topology-Geometry 'topo_schema_to_create' (id:7) created.
```




[TopoGeo_LoadGeometry](#), [AddTopoGeometryColumn](#), [CreateTopology](#), [DropTopology](#)

9.6.5 TopoGeo_AddPoint

`TopoGeo_AddPoint` —  (split) .

Synopsis

`bigint TopoGeo_AddPoint`(varchar atopolology, 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](#)

9.6.6 TopoGeo_AddLineString

`TopoGeo_AddLineString` — Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.

Synopsis

`SETOF bigint TopoGeo_AddLineString`(varchar atopolology, geometry aline, float8 tolerance, int max_edges);



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, in addition to new edges.

The returned edge identifiers may be either existing edges or newly created edges as needed to fully represent the input line as closely as possible.

The number of newly created edges (either covering space previously uncovered or resulting from split of existing edges) may be limited by the `max_edges` parameter.



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 简体中文。

Enhanced: 3.2.0 added support for returning signed identifier.

Enhanced: 3.7.0 added support for limiting the number of new edges created in the topology.

☐☐

[TopoGeo_AddPoint](#), [TopoGeo_AddPolygon](#), [TopoGeo_LoadGeometry](#), [AddEdge](#), [CreateTopology](#)

9.6.7 TopoGeo_AddPolygon

`TopoGeo_AddPolygon` — Adds a polygon to an existing topology using a tolerance and possibly splitting existing edges/faces. Returns face identifiers.

Synopsis

SETOF bigint **TopoGeo_AddPolygon**(varchar atopology, geometry apoly, float8 tolerance);

☐☐

Adds a polygon to an existing topology and returns a set of face identifiers forming it up. The boundary of the given polygon will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the boundary of the new polygon.



Note

Updating statistics about topologies being loaded via this function is up to caller, see [maintaining statistics during topology editing and population](#).

2.0.0 简体中文。

☐☐

[TopoGeo_AddPoint](#), [TopoGeo_AddLineString](#), [TopoGeo_LoadGeometry](#), [AddFace](#), [CreateTopology](#)

9.6.8 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);

函数

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

函数

[TopoGeo_AddPoint](#), [TopoGeo_AddLineString](#), [TopoGeo_AddPolygon](#), [CreateTopology](#)

9.7 函数

9.7.1 ST_AddIsoNode

ST_AddIsoNode — 添加 (isolated) 节点 ID 到拓扑。如果 NULL 则失败，返回失败消息。

Synopsis

bigint **ST_AddIsoNode**(varchar atopolgy, bigint aface, geometry apoint);

参数

atopolgy 拓扑 aface ID(faceid) 拓扑 ID 节点 ID 添加 ID(nodeid) 失败。

如果 (SRID) 不为空, apoint 不为空, 则 NULL 失败, 返回失败消息 (失败消息) 失败。失败。

aface 不为 NULL 失败 apoint 不为空, 失败。

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X+1.3.1

函数

函数

[AddNode](#), [CreateTopology](#), [DropTopology](#), [ST_Intersects](#)

9.7.2 ST_AddIsoEdge

`ST_AddIsoEdge` — 将 `anode` 与 `anothernode` 通过 `alinestring` 添加为孤立边，并返回边的 ID。

Synopsis

`bigint ST_AddIsoEdge`(`varchar` `atopology`, `bigint` `anode`, `bigint` `anothernode`, `geometry` `alinestring`);

返回

返回 `anode` 与 `anothernode` 通过 `alinestring` 添加的孤立边的 ID (edgeid)。

`alinestring` 的 SRID 不为 NULL，且 `alinestring` 必须与 `anode` 和 `anothernode` 具有相同的 SRID。

`alinestring` 必须与 `anode` 和 `anothernode` 具有相同的 SRID。

`anode` 与 `anothernode` 必须与 `alinestring` 具有相同的 SRID。

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](#)

9.7.3 ST_AddEdgeNewFaces

`ST_AddEdgeNewFaces` — 将 `acurve` 添加到 `atopology` 中，并返回新面的 ID。

Synopsis

`bigint ST_AddEdgeNewFaces`(`varchar` `atopology`, `bigint` `anode`, `bigint` `anothernode`, `geometry` `acurve`);

返回

返回 `acurve` 添加到 `atopology` 中后，新面的 ID。

返回 `acurve` 添加到 `atopology` 中后，新面的 ID。

`acurve` 必须与 `anode` 和 `anothernode` 具有相同的 SRID。如果 `acurve` 为 NULL，则返回 `anode` 和 `anothernode` 之间的边。

`acurve` 的 SRID 不为 NULL。

2.0 版本。

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.12

返回

返回

[ST_RemEdgeNewFace](#)

[ST_AddEdgeModFace](#)

9.7.4 ST_AddEdgeModFace

`ST_AddEdgeModFace` — 添加边到面，并修改面的拓扑，并返回新的面。

Synopsis

`bigint ST_AddEdgeModFace(varchar atopology, bigint anode, bigint anothernode, geometry acurve);`

返回

返回类型，返回类型，返回类型。



Note

添加边，并修改面的拓扑。返回类型 (面) 返回类型 (universe face) 返回类型。

返回类型 ID 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 node 返回类型 返回类型), `acurve` 返回类型 LINESTRING 返回类型, `anode` 返回类型 `anothernode` 返回类型 `acurve` 返回类型。

`acurve` 返回类型 (SRID) 返回类型。

2.0 返回类型。



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13

返回

返回

[ST_RemEdgeModFace](#)

[ST_AddEdgeNewFaces](#)

9.7.5 ST_RemEdgeNewFace

`ST_RemEdgeNewFace` — 从面中移除边，并返回新的面。

Synopsis

bigint **ST_RemEdgeNewFace**(varchar atopology, bigint anedge);

00000000, 000000000000000000000000, 000000000000000000000000.
 00000000 ID 000000, 0000000000000000 NULL 000000. 0000000000
 00000000, 000000, 00 (0000000000000000000000) 000000000000
 000000.
 00000000000000000000000000000000.

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 u 是 NULL 指针, 那么 u 是空边 (即 u 是空边 u 的父节点), 那么 u 是空边 u 的父节点。

2.0 ☒☒☒☒☒☒☒☒☒☒.



✔ This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14



ST RemEdgeModFace

ST AddEdgeNewFaces

9.7.6 ST_RemEdgeModFace

ST_RemEdgeModFace — Removes an edge, and if the edge separates two faces deletes one face and modifies the other face to cover the space of both.

Synopsis

bigint **ST_RemEdgeModFace**(varchar atopology, bigint anedge);

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.

[illegible]

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).

If `edge` is `NULL`, the edge is not found. If `edge` is not `NULL`, the edge is found.

2.0 ☒☒☒☒☒☒☒☒☒☒.



✔ This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.15



ST_AddEdgeModFace

ST_RemEdgeNewFace

9.7.7 ST_ChangeEdgeGeom

ST_ChangeEdgeGeom — .

Synopsis

```
text ST_ChangeEdgeGeom(varchar atopology, bigint anedge, geometry acurve);
```

[illegible]

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 0000000000000000 (SRID) 0000000000000000.

acurve $\square\square\square\square\square\square\square\square\square$, $\square\square\square\square\square\square\square\square$.

[illegible]

1.1.0 ☒☒☒☒☒☒☒☒☒☒☒.

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

✔ 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

9.7.8 ST_ModEdgeSplit

ST_ModEdgeSplit — Splits an edge in a topology, adding a new node at the specified location. Returns the id of the new node.

Synopsis

bigint ST_ModEdgeSplit(varchar atopology, bigint anedge, geometry apoint);

ST_ModEdgeSplit

ST_ModEdgeSplit splits an edge in a topology, adding a new node at the specified location. Returns the id of the new node. The new node is added to the topology and the edge is split into two edges. The original edge is returned as the first edge and the new edge is returned as the second edge.

Availability: 1.1

Changed: 2.0 **ST_ModEdgesSplit**, **ST_ModEdgesSplit** **ST_ModEdgesSplit**.

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

ST_ModEdgeSplit

Add an edge to split.

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227592 893910, 227600 893910)', 26986)) AS edgeid;
```

3

Split the edge with a new node.

```
SELECT topology.ST_ModEdgeSplit('ma_topo', 3, ST_SetSRID(ST_Point(227594, 893910), 26986)) AS node_id;
```

7

ST_ModEdgeSplit

ST_NewEdgesSplit, **ST_ModEdgeHeal**, **ST_NewEdgeHeal**, **AddEdge**

9.7.9 ST_ModEdgeHeal

ST_ModEdgeHeal — Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node.

Synopsis

bigint ST_ModEdgeHeal(varchar atopology, bigint anedge, bigint anotheredge);

☒☒

Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node. Updates all existing joined edges and relationships accordingly.

2.0 ☒☒☒☒☒☒☒☒☒☒☒☒.

✔ This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

☒☒

[ST_ModEdgeSplit](#) [ST_NewEdgesSplit](#)

9.7.10 ST_NewEdgeHeal

ST_NewEdgeHeal — Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided.

Synopsis

bigint **ST_NewEdgeHeal**(varchar atopology, bigint anedge, bigint anotheredge);

☒☒

Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided. Returns the id of the new edge replacing the healed ones. Updates all existing joined edges and relationships accordingly.

2.0 ☒☒☒☒☒☒☒☒☒☒☒☒.

✔ This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

☒☒

[ST_ModEdgeHeal](#) [ST_ModEdgeSplit](#) [ST_NewEdgesSplit](#)

9.7.11 ST_MoveIsoNode

ST_MoveIsoNode — Moves an isolated node in a topology from one point to another. If new apoint geometry exists as a node an error is thrown. Returns description of move.

Synopsis

text **ST_MoveIsoNode**(varchar atopology, bigint anode, geometry apoint);

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX. **apoint** XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXX.

If any arguments are null, the `apoint` is not a point, the existing node is not isolated (is a start or end point of an existing edge), new node location intersects an existing edge (even at the end points) or the new location is in a different face (since 3.2.0) then an exception is thrown.

XXXXXXXXXXXXXXXXXXXXXXXXXXXX (SRID) XXXXXXXXXXXXXXXXXXXXXXXX.

2.0.0 □□□□□□□□□□.

Enhanced: 3.2.0 ensures the node cannot be moved in a different face



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.2



Add an isolated node with no face.

```
SELECT topology.ST_AddIsoNode('ma_topo', NULL, ST_GeomFromText('POINT(227579 893916)', 26986)) AS nodeid;
```

7

Move the new node.

```
SELECT topology.ST_MoveIsoNode('ma_topo', 7, ST_GeomFromText('POINT(227579.5 893916.5)', 26986)) AS descrip;
```

Isolated Node 7 moved to location 227579.5,893916.5



ST AddIsoNode

9.7.12 ST NewEdgesSplit

ST_NewEdgesSplit — $\{0, 1\}^{\text{ST\_NewEdgesSplit}}$, $\{0, 1\}^{\text{ST\_NewEdgesSplit} \times 2}$ $\{0, 1\}^{\text{ST\_NewEdgesSplit}}$. $\{0, 1\}^{\text{ST\_NewEdgesSplit}}$ ID $\{0, 1\}^{\text{ST\_NewEdgesSplit}}$.

Synopsis

bigint **ST_NewEdgesSplit**(varchar atopology, bigint anedge, geometry apoint);[illegible]

`ST_AddEdgeToFace` (SRID) 整数, `apoint` 点, 如果为 NULL 则为面中的第一个点, 如果为 NULL 则为面中的最后一个点, 如果为 NULL 则为面中的第一个点, 如果为 NULL 则为面中的最后一个点。

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.8

添加

Add an edge to split.

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575 893917, 227592 893900)', 26986)) AS edgeid;
```

2

Split the new edge.

```
SELECT topology.ST_NewEdgesSplit('ma_topo', 2, ST_GeomFromText('POINT(227578.5 893913.5)', 26986)) AS newnodeid;
```

6

添加

[ST_ModEdgeSplit](#) [ST_ModEdgeHeal](#) [ST_NewEdgeHeal](#) [AddEdge](#)

9.7.13 ST_RemoveIsoNode

`ST_RemoveIsoNode` — 移除拓扑中的孤立节点。如果节点是孤立节点 (没有面与之关联), 则返回 true, 否则返回 false。

Synopsis

text **ST_RemoveIsoNode**(varchar atopology, bigint anode);

添加

移除拓扑中的孤立节点。如果节点是孤立节点 (没有面与之关联), 则返回 true, 否则返回 false。

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

添加

Remove an isolated node with no face.

```
SELECT topology.ST_RemoveIsoNode('ma_topo', 7) AS result;
```

Isolated node 7 removed



[ST_AddIsoNode](#)

9.7.14 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, bigint anedge);



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



Remove an isolated edge with no face.

```
SELECT topology.ST_RemoveIsoEdge('ma_topo', 7) AS result;
```

```
Isolated edge 7 removed
```



[ST_AddIsoNode](#)

9.8

9.8.1 GetEdgeByPoint

GetEdgeByPoint — Finds the edge-id of an edge that intersects a given point.

Synopsis

bigint **GetEdgeByPoint**(varchar atopology, geometry apoint, float8 tol1);

函数

Retrieves the id of an edge that intersects a Point.

`GetEdgeById(geometry topology, geometry point, integer edgeid) integer`. tolerance = 0 返回一个整数。

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

当 tolerance = 0 时，`ST_Intersects` 与 `ST_DWithin` 类似。

GEOS 版本

2.0.0 版本。

函数

函数 `AddEdge` 用于添加边。

```
SELECT topology.GetEdgeByPoint('ma_topo', geom, 1) As with1mtol, topology.GetEdgeByPoint('ma_topo', geom, 0) As withnotol
FROM ST_GeomFromEWKT('SRID=26986;POINT(227622.6 893843)') As geom;
```

with1mtol	withnotol
2	0

This query errors because two or more edges are found.

```
SELECT topology.GetEdgeByPoint('ma_topo', geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

ERROR: Two or more edges found

函数

`AddEdge`, `GetNodeByPoint`, `GetFaceByPoint`

9.8.2 GetFaceByPoint

`GetFaceByPoint` — Finds face intersecting a given point.

Synopsis

`bigint 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
```

This query errors because two or more faces are found.

```
SELECT topology.GetFaceByPoint('ma_topo', geom, 1) As nearnode
FROM ST_GeomFromEWKT('POINT(227591.9 893900.4)') As geom;
```

```
ERROR: Two or more faces found
```

国国

[GetFaceContainingPoint](#), [AddFace](#), [GetNodeByPoint](#), [GetEdgeByPoint](#)

9.8.3 GetFaceContainingPoint

GetFaceContainingPoint — Finds the face containing a point.

Synopsis

bigint **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](#)

9.8.4 GetNodeByPoint

GetNodeByPoint — Finds the node-id of a node at a point location.

Synopsis

bigint **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;
```

2

This query errors because the tolerance matches two or more nodes.

```
SELECT topology.GetNodeByPoint('ma_topo', geom, 1000) As too_much_tolerance
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

ERROR: Two or more nodes found

☒☒

[AddEdge](#), [GetEdgeByPoint](#), [GetFaceByPoint](#)

9.8.5 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;
```

```
1
```

返回

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologySRID](#)

9.8.6 GetTopologySRID

GetTopologySRID — 返回 topology.topology 的 SRID。

Synopsis

integer **GetTopologyID**(varchar toponame);

返回

返回 topology.topology 的 SRID。

2.0.0 版本引入。

返回

```
SELECT topology.GetTopologySRID('ma_topo') AS srid;
```

```
4326
```

返回

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologyID](#)

9.8.7 GetTopologyName

GetTopologyName — 返回 ID 指定的拓扑名称。

Synopsis

varchar **GetTopologyName**(integer topology_id);

返回

返回 ID 指定的 topology.topology 名称。

Availability: 1.1

示例

```
SELECT topology.GetTopologyName(1) AS topo_name;
```

```
ma_topo
```

示例

[CreateTopology](#), [DropTopology](#), [GetTopologyID](#), [GetTopologySRID](#)

9.8.8 ST_GetFaceEdges

ST_GetFaceEdges — 返回面 ID 指定的面边缘。

Synopsis

getfaceedges_returntype **ST_GetFaceEdges**(varchar atopology, bigint aface);

返回

aface 指定的面 ID。返回 (sequence) ID(edgeid) 列表。从 1 开始。

返回的列表包含面 ID 指定的面边缘 ID。返回的列表包含面 ID 指定的面边缘 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

¶¶

This query returns the edges that bound face 1.

```
SELECT (topology.ST_GetFaceEdges('tt', 1)).*;
```

sequence	edge
1	-4
2	5
3	7
4	-6
5	1
6	2
7	3

This query returns the sequence, edge identifier, and geometry of the edges that bound face 1. Use [ST_GetFaceGeometry](#) if only the geometry and sequence are needed.

```
SELECT t.seq, t.edge, geom
FROM topology.ST_GetFaceEdges('tt', 1) As t(seq, edge)
      INNER JOIN tt.edge AS e ON abs(t.edge) = e.edge_id;
```

¶¶

[GetRingEdges](#), [AddFace](#), [ST_GetFaceGeometry](#)

9.8.9 ST_GetFaceGeometry

`ST_GetFaceGeometry` — ¶¶¶¶¶¶¶¶¶¶ ID ¶¶¶¶¶¶¶¶¶¶.

Synopsis

geometry **ST_GetFaceGeometry**(varchar atopology, bigint aface);

¶¶

¶¶¶¶¶¶¶¶¶¶ 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

¶¶

This query returns the WKT of the polygon added with [AddFace](#).

```
SELECT ST_AsText(topology.ST_GetFaceGeometry('ma_topo', 1)) AS facegeomwkt;
```

```
POLYGON((234776.9 899563.7,234896.5 899456.7,234914 899436.4,234946.6 899356.9,
234872.5 899328.7,234891 899285.4,234992.5 899145,234890.6 899069,
234755.2 899255.4,234612.7 899379.4,234776.9 899563.7))
```

返回

AddFace

9.8.10 GetRingEdges

GetRingEdges — 返回指定面环的边。

Synopsis

```
getfaceedges_returntype GetRingEdges(varchar atopology, bigint aring, integer max_edges=null);
```

返回

返回指定面环的边。返回结果是一个表，包含以下列：ID (sequence) 和 ID(edgeid)。返回结果按 ID 升序排列。返回结果最多包含 1 条边。

返回结果包含 ID 列，返回结果按 ID 升序排列。返回结果最多包含 1 条边。返回结果包含 ID 列，返回结果按 ID 升序排列。返回结果最多包含 1 条边。

max_edges 为 NULL 时返回所有边。返回结果按 ID 升序排列。返回结果最多包含 1 条边。



Note

返回结果按 ID 升序排列。

2.0.0 版本引入。

返回

ST_GetFaceEdges, GetNodeEdges

9.8.11 GetNodeEdges

GetNodeEdges — 返回指定节点的边。

Synopsis

```
getfaceedges_returntype GetNodeEdges(varchar atopology, bigint anode);
```

返回

返回指定节点的边。返回结果是一个表，包含以下列：ID 和 ID(edgeid)。返回结果按 ID 升序排列。返回结果最多包含 1 条边。返回结果包含 ID 列，返回结果按 ID 升序排列。返回结果最多包含 1 条边。

**Note**

PostGIS 3.7.0alpha1 简体中文

2.0 简体中文。

简体中文

[getfaceedges_returntype](#), [GetRingEdges](#), [ST_Azimuth](#)

9.8.12 FindVertexSegmentPairsBelowDistance

FindVertexSegmentPairsBelowDistance — Find pairs of topology vertex/segment that are closer than tolerated distance

Synopsis

setof record **FindVertexSegmentPairsBelowDistance**(varchar atopology, float8 tolerance);

简体中文

Returns a relation in which each tuple is a pair of vertex/segment where the vertex is NOT an endpoint of the segment but the distance between them is below the given tolerance.

Availability: 3.7

简体中文

[ValidateTopologyPrecision](#)

9.9 简体中文

9.9.1 Polygonize

Polygonize — Finds and registers all faces defined by topology edges.

Synopsis

text **Polygonize**(varchar toponame);

注册

Registers all faces that can be built out a topology edge primitives.

注册所有可以构建拓扑边原型的面。



Note

注册所有可以构建拓扑边原型的面，`Polygonize` 注册所有可以构建拓扑边原型的边。



Note

注册边 `next_left_edge` 和 `next_right_edge` 注册所有可以构建拓扑边原型的边。

2.0.0 注册所有可以构建拓扑边原型的边。

注册

AddFace, ST_Polygonize

9.9.2 AddNode

AddNode — 注册所有可以构建拓扑边原型的边 `ID(nodeid)` 注册所有可以构建拓扑边原型的边，注册 `ID` 注册所有可以构建拓扑边原型的边。

Synopsis

`bigint AddNode(varchar toponame, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);`

注册

注册所有可以构建拓扑边原型的边。 **AddEdge** 注册所有可以构建拓扑边原型的边，注册所有可以构建拓扑边原型的边。

注册所有可以构建拓扑边原型的边，`allowEdgeSplitting` 注册所有可以构建拓扑边原型的边。

`computeContainingFace` 注册所有可以构建拓扑边原型的边。



Note

`apoint` 注册所有可以构建拓扑边原型的边，注册所有可以构建拓扑边原型的边 `ID(nodeid)` 注册所有可以构建拓扑边原型的边。

2.0.0 注册所有可以构建拓扑边原型的边。

例

```
SELECT topology.AddNode('ma_topo', ST_GeomFromText('POINT(227641.6 893816.5)', 26986)) AS  
    nodeid;
```

4

例

AddEdge, CreateTopology

9.9.3 AddEdge

AddEdge — 向拓扑中添加边，并返回新边的 ID(edgeid)。

Synopsis

bigint **AddEdge**(varchar toponame, geometry aline);

例

向名为 toponame 的拓扑中添加边，并返回新边的 ID(edgeid)。该边是“宇宙”(universe)的一部分。



Note

aline 必须是二维的，并且是有效的几何体。



Note

aline 的 srid 必须与 toponame 的 srid 相同。

GEOS 依赖



Warning

AddEdge 已弃用，自 3.5.0 起。请使用 **TopoGeo_AddLineString** 代替。

2.0.0 版本已弃用。

❏

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575.8 893917.2, 227591.9 893900.4)', 26986)) AS 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;
```

2

This query errors because the edge intersects an existing edge away from the endpoints.

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.2 893900, 227591.9 893900.4, 227704.5 893778.5)', 26986)) AS edgeid;
```

```
ERROR: Edge intersects (not on endpoints) with existing edge 1
```

❏

[TopoGeo_AddLineString](#), [CreateTopology](#), Section 4.5

9.9.4 AddFace

AddFace — 面元 (face primitive) 的函数。

Synopsis

bigint **AddFace**(varchar toponame, geometry apolygon, boolean force_new=false);

❏

面元 (face primitive) 的函数。

left face 或 right face 面元。面元包含在指定的面元中。面元包含在指定的面元中。



Note

面元 edge 的 next_left_edge 或 next_right_edge 面元。

面元 (面元) 的函数。面元包含在指定的面元中。

apolygon 面元, force_new 或 (面元) 面元 ID 面元, force_new 面元 ID 面元。

**Note**

ST_MakeLine (force_new = true) 选项, 该选项, 强制将输入点集转换为线。如果输入点集是 MBR 则返回 MBR。否则, 返回由点集定义的线。

**Note**

apolygon 函数 srid 参数指定输出 srid 值。如果未指定, 则使用输入几何体的 srid 值。

2.0.0 版本中, 该函数已弃用。

❗

First add the edges using `generate_series` as an iterator. This query only works for polygons with fewer than 10000 points because of the maximum value in `generate_series`.

```
SELECT topology.AddEdge('ma_topo', ST_MakeLine(ST_PointN(geom,i), ST_PointN(geom, i + 1) )) ←
  As edgeid
FROM (SELECT ST_NPoints(geom) AS npt, geom
      FROM
        (SELECT ST_Boundary(ST_GeomFromText('POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5 899328.7,
          234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
          234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) ) As geom
        ) As geoms) As facen CROSS JOIN generate_series(1,10000) As i
WHERE i < npt;
```

3
4
5
6
7
8
9
10
11
12

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;
```

1

❗

[AddEdge](#), [CreateTopology](#), [Section 4.5](#)

9.9.5 ST_Simplify

ST_Simplify — 使用 Douglas-Peucker 算法对 TopoGeometry 对象进行简化。

Synopsis

```
geometry ST_Simplify(topogeometry tg, float8 tolerance);
```

注意

使用 Douglas-Peucker 算法对 TopoGeometry 对象进行简化。



Note

简化后的 TopoGeometry 对象可能包含无效的几何体。使用 ST_IsValid 检查。

GEOS 版本

2.1.0 及以上版本。

注意

ST_Simplify, ST_IsSimple, ST_IsValid, ST_ModEdgeSplit

9.9.6 RemoveUnusedPrimitives

RemoveUnusedPrimitives — 移除拓扑原语，这些原语不需要来定义现有的 TopoGeometry 对象。

Synopsis

```
bigint RemoveUnusedPrimitives(text topology_name, geometry bbox);
```

注意

找到所有原语（节点、边、面）这些原语不是严格需要来代表现有的 TopoGeometry 对象并移除它们，保持拓扑有效性（边链接、面标注）和 TopoGeometry 空间占用。

不会创建新的原语标识符，而是扩展现有的原语来包括合并的面（在移除边后）或愈合的边（在移除节点后）。

可用性: 3.3.0

注意

ST_ModEdgeHeal, ST_RemEdgeModFace

9.10 TopoGeometry 函数

9.10.1 CreateTopoGeom

CreateTopoGeom — 在指定的拓扑层中创建 TopoGeometry 对象。tg_type 为 1: [点] 对象, 2: [线] 对象, 3: [面] 对象, 4: 集合对象。

Synopsis

```
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id, topoelementarray tg_objs, bigint tg_id);
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id);
```

函数

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.

该函数用于在指定的拓扑层中创建 TopoGeometry 对象，并将其注册到拓扑关系表中。该函数接受以下参数：

toponame: 拓扑名称。layer_id: 拓扑层 ID。tg_type: 拓扑对象类型。tg_objs: 拓扑对象数组。tg_id: 拓扑对象 ID。

Availability: 1.1

示例: 创建 TopoGeometry 对象

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);
```

函数: 在 TopoGeometry 对象中添加边

该函数用于在指定的 TopoGeometry 对象中添加边。该函数接受以下参数：

Create the topogeometry column.

```
SELECT topology.AddTopoGeometryColumn(
    'topo_boston',
    'boston', 'blockgroups', 'topo', 'POLYGON');
```

1

Update the column assuming everything is perfectly aligned with the 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, so this version allows some error. It counts a face if 50 percent of it falls within the expected block group 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;

```

To convert the topogeometry back to a denormalized geometry aligned with faces and edges, cast the topogeometry to a geometry. The resulting geometries are aligned with the TIGER street centerlines.

```
UPDATE boston.blockgroups SET new_geom = topo::geometry;
```

囧囧

[AddTopoGeometryColumn](#), [toTopoGeom](#) [ST_CreateTopoGeo](#), [ST_GetFaceGeometry](#), [TopoElementArray](#), [TopoElementArray_Agg](#)

9.10.2 toTopoGeom

toTopoGeom — Converts a simple Geometry into a topo geometry.

Synopsis

topogeometry **toTopoGeom**(geometry geom, varchar toponame, integer layer_id, float8 tolerance=-1);

topogeometry **toTopoGeom**(geometry geom, topogeometry topogeom, float8 tolerance=-1);



TopoGeometry.

000000000000000000000000000000000000. 000000000000000000000000000000000000.
 00000000 relation 00000000 TopoGeometry 00000000.

TopoGeometry (topogeom) is a package.

When tolerance is given it will be used to snap the input geometry to existing primitives. The special value -1 (default) means use the topology precision as tolerance.

1 (toponame) (layer_id) TopoGeometry .

2 次, TopoGeometry(toponame) を再帰的に呼び出す。TopoGeometry
オブジェクトを再帰的に呼び出す。clearTopoGeom を再帰的に
呼び出す。

2.0 ☒☒☒☒☒☒☒☒☒☒.

TopoGeometry 2.1.0

Changed: 3.7.0 changes meaning of tolerance 0 to really mean 0 and -1 (new default) to request using the topology precision.

□□□□□□□□□□□□□□□□□□ (workflow) □□□.

Create the topology without 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 the new table.

```
SELECT topology.AddTopoGeometryColumn('topo_boston_test', 'public', 'nei_topo', 'topo', ' ←
MULTIPOLYGON') AS new_layer_id;
```

1

Use the new layer identifier to populate the topogeometry column.

```
INSERT INTO nei_topo(nei, topo)
SELECT nei, topology.toTopoGeom(geom, 'topo_boston_test', 1)
FROM neighborhoods
WHERE gid BETWEEN 1 and 15;
```

Verify what happened.

```
SELECT * FROM
    topology.TopologySummary('topo boston test');
```

```
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);
```

Find faces that are left without any TopoGeometry reference after the shrink operation.

```
SELECT ST_GetFaceGeometry('topo_boston_test', f.face_id)
FROM topo_boston_test.face f
WHERE f.face_id
> 0
AND NOT EXISTS (
  SELECT * FROM topo_boston_test.relation
  WHERE layer_id = 1 AND element_id = f.face_id
);
```

☐☐

[CreateTopology](#), [AddTopoGeometryColumn](#), [CreateTopoGeom](#), [TopologySummary](#), [clearTopoGeom](#)

9.10.3 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] ORDER BY e, t) As tea
FROM generate_series(1,3) As e CROSS JOIN generate_series(1,4) As t;
```

```
{{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](#)

9.10.4 TopoElement

TopoElement — Converts a topogeometry to a topoelement.

Synopsis

topoelement **TopoElement**(topogeometry topo);

返回

Converts a **TopoGeometry** to a **TopoElement**.

Availability: 3.4.0

返回

返回每个topogeometry的TopoElement表示 (workflow) 返回。

Return the TopoElement representation of each topogeometry.

```
SELECT TopoElement(topo)
FROM neighborhoods;
```

Use a cast to aggregate topogeometries into a TopoElementArray.

```
SELECT topology.TopoElementArray_Agg(topo::topoelement)
FROM neighborhoods
GROUP BY city;
```

返回

TopoElementArray_Agg, **TopoGeometry**, **TopoElement**

9.11 TopoGeometry 函数

9.11.1 clearTopoGeom

clearTopoGeom — Clears the content of a topo geometry.

Synopsis

topogeometry **clearTopoGeom**(topogeometry topogeom);

返回

TopoGeometry 函数返回 **TopoGeometry** 类型。 **toTopoGeom** 函数返回 **TopoGeometry** 类型。

2.1 返回。

返回

Shrink all TopoGeometry polygons by 10 meters.

```
UPDATE nei_topo SET topo = toTopoGeom(ST_Buffer(topo, -10), clearTopoGeom(topo), 0);
```


☒☒

[TopoGeom_addElement](#), [CreateTopoGeom](#)

9.11.4 TopoGeom_addTopoGeom

`TopoGeom_addTopoGeom` — Adds element of a TopoGeometry to the definition of another TopoGeometry.

Synopsis

topogeometry **TopoGeom_addTopoGeom**(topogeometry tgt, topogeometry src);

☒☒

Adds the elements of a [TopoGeometry](#) to the definition of another TopoGeometry, possibly changing its cached type (type attribute) to a collection, if needed to hold all elements in the source object.

The two TopoGeometry objects need be defined against the *same* topology and, if hierarchically defined, need be composed by elements of the same child layer.

Availability: 3.2

☒☒

Set an overall TopoGeometry value to be composed of all elements from specific TopoGeometry values.

```
UPDATE mylayer SET tg_overall = TopoGeom_addTopoGeom(
    TopoGeom_addTopoGeom(
        clearTopoGeom(tg_overall),
        tg_specific1
    ),
    tg_specific2
);
```

☒☒

[TopoGeom_addElement](#), [clearTopoGeom](#), [CreateTopoGeom](#)

9.11.5 toTopoGeom

`toTopoGeom` — Adds a geometry shape to an existing topo geometry.

☒☒

Refer to [toTopoGeom](#).

9.12 TopoGeometry 函数

9.12.1 GetTopoGeomElementArray

GetTopoGeomElementArray — Returns a `topoelementarray` (an array of `topoelements`) containing the topological elements and type of the given `TopoGeometry` (primitive elements).

Synopsis

```
topoelementarray GetTopoGeomElementArray(varchar toponame, integer layer_id, bigint tg_id);
topoelementarray GetTopoGeomElementArray(topogeometry tg);
```

返回

`topoelementarray` `TopoGeometry` 对象 (对象ID) 返回 `TopoElementArray` 对象。返回的 `topoelementarray` 包含 `GetTopoGeomElements` 返回的 `topoelements`。

`tg_id` 或 `topology.layer` 或 `layer_id` 或 `TopoGeometry` 的 `TopoGeometry` ID。

Availability: 1.1

返回

返回

`GetTopoGeomElements`, `TopoElementArray`

9.12.2 GetTopoGeomElements

GetTopoGeomElements — Returns a set of `topoelement` objects containing the topological `element_id`, `element_type` of the given `TopoGeometry` (primitive elements).

Synopsis

```
setof topoelement GetTopoGeomElements(varchar toponame, integer layer_id, bigint tg_id);
setof topoelement GetTopoGeomElements(topogeometry tg);
```

返回

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` 或 `topology.layer` 或 `layer_id` 或 `TopoGeometry` 的 `TopoGeometry` ID。

2.0.0 版本引入。

[GetTopoGeomElementArray](#), [TopoElement](#), [TopoGeom_addElement](#), [TopoGeom_remElement](#)

9.12.3 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));
         4326
```

Section [4.5](#), [ST_SetSRID](#), [ST_Transform](#), [ST_SRID](#)

9.13 TopoGeometry

9.13.1 AsGML

AsGML — TopoGeometry  GML       .


```

>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
    384833 236884 384844 236882 384866 236881 384879 236883 384954 ←
    236898 385087 236932 385117 236938
    385167 236938 385203 236941 385224 236946 385233 236950 385241 ←
    236956 385254 236971
    385260 236979 385268 236999 385273 237018 385273 237037 385271 ←
    237047 385267 237057 385225 237125
    385210 237144 385192 237161 385167 237192 385162 237202 385159 ←
    237214 385159 237227 385162 237241
    385166 237256 385196 237324 385209 237345 385234 237375 385237 ←
    237383 385238 237399 385236 237407
    385227 237419 385213 237430 385193 237439 385174 237451 385170 ←
    237455 385169 237460 385171 237475
    385181 237503 385190 237521 385200 237533 385206 237538 385213 ←
    237541 385221 237542 385235 237540 385242 237541
    385249 237544 385260 237555 385270 237570 385289 237584 385292 ←
    237589 385291 237596 385284 237630</gml:posList>
    </gml:LineStringSegment>
  </gml:segments>
</gml:Curve>
</gml:curveProperty>
</gml:Edge>
</gml:directedEdge>
</gml:TopoCurve>

```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.

```

SELECT topology.AsGML(topo,'') As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';

```

```

<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>
      </directedNode>
    </Edge>
  </directedEdge>
</TopoCurve>

```

```

        </segments>
      </Curve>
    </curveProperty>
  </Edge>
</directedEdge>
</TopoCurve>

```



CreateTopoGeom, ST_CreateTopoGeo

9.13.2 AsTopoJSON

AsTopoJSON — TopoGeometry ☒ TopoJSON ☒☒☒☒☒☒☒☒☒.

Synopsis

```
text AsTopoJSON(topogeometry tg, regclass edgeMapTable);
```

[illegible]

XXXXXXXXXX, XXXX" XX (serial)" XX"arc_id" XXXXXXXX"edge_id" XXXXXXXXXX.
 XXX"edge id" XXXXXXXXXX



Note

TopoJSON 0-1 "edgeMapTable" 1-1.

TopoJSON 是, 是 (snippet), 是, 是
TopoJSON 是。

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.

□□□□: 2.2.1 □□□□□□ (puntal) □□□□□□□□□□□□.



ST AsGeoJSON



Create a temporary edge map for the TopoJSON output.

```
CREATE TEMP TABLE edgemap(arc_id serial, edge_id int unique);
```

Build the TopoJSON header, objects, arcs, and footer.

```

SELECT '{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects" ←
": { '

UNION ALL SELECT '' || feature_name || ': ' || AsTopoJSON(feature, 'edgemap')
FROM features.big_parcel WHERE feature_name = 'P3P4';

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

UNION ALL SELECT '}]':::text as t;

{ "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]]
]]
}

```

9.14 拓扑几何

9.14.1 Equals

Equals — 检查 TopoGeometry 是否相等。

Synopsis

boolean **Equals**(topogeometry tg1, topogeometry tg2);

[illegible]

Note

TopoGeometry. TopoGeometry.

1.1.0 ☒☒☒☒☒☒☒☒☒☒☒.



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



The following example uses the `nei_topo` table created in the `toTopoGeom` examples.

```
SELECT topology.Equals(a.topo, b.topo)
FROM nei_topo AS a
JOIN nei_topo AS b ON b.gid = a.gid
WHERE a.gid = 1;
```

t

GetTopoGeomElements, ST_Equals

9.14.2 Intersects

Intersects — ☒ TopoGeometry ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

Synopsis

```
boolean Intersects(topogeometry tq1, topogeometry tq2);
```

[illegible]

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.

☒☒

The following example uses the `nei_topo` table created in the [toTopoGeom](#) examples.

```
SELECT topology.Intersects(a.topo, b.topo)
FROM nei_topo AS a
JOIN nei_topo AS b ON b.gid = a.gid
WHERE a.gid = 1;
```

t

☒☒

[ST_Intersects](#)

9.15 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
```

9.15.1 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.

9.15.2 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 的一个可执行文件。

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.

UNIX 安装和配置指南:

```
raster2pgsql -s 4326 -I -C -M *.tif -F -t 100x100 public.demelevation | psql -d gisdb
```

安装和配置指南, 安装 2 到 4 个
安装和配置指南, 安装和配置指南 (安装和配置指南 DB 安装), 安装和配置指南
安装和配置指南 -e 安装和配置指南 (安装和配置指南安装和配置指南安装和配置指南
安装和配置指南). 安装和配置指南 128x128 安装和配置指南安装和配置指南安装和配置指南
安装和配置指南. 安装和配置指南安装和配置指南安装和配置指南 -F 安装和配置指南“filename” 安装和配置指南
安装和配置指南.

```
raster2pgsql -I -C -e -Y -F -s 26986 -t 128x128 -l 2,4 bostonaerials2008/*.jpg aeriels. ↵  
boston | psql -U postgres -d gisdb -h localhost -p 5432
```

Get a list of raster types supported:

```
raster2pgsql -G
```

-G 安装和配置指南:

Available GDAL raster formats:

```
Virtual Raster  
GeoTIFF  
National Imagery Transmission Format  
Raster Product Format TOC format  
ECRG TOC format  
Erdas Imagine Images (.img)  
CEOS SAR Image  
CEOS Image  
...  
Arc/Info Export E00 GRID  
ZMap Plus Grid  
NOAA NGS Geoid Height Grids
```

10.1.1.2 raster2pgsql options

Long options that take values also accept the `--option=value` spelling.

-?, --help 安装和配置指南. 安装和配置指南.

-G, --gdal-formats 安装和配置指南.

c|a|d|p -- 安装和配置指南:

```
-c 安装和配置指南 (安装和配置指南) 安装和配置指南. 安装和配置指南.  
-a 安装和配置指南 (安装和配置指南) 安装和配置指南.  
-d 安装和配置指南, 安装和配置指南 (安装和配置指南) 安装和配置指南.  
-p 安装和配置指南, 安装和配置指南.
```

--if-not-exists Use IF NOT EXISTS for table creation in -c and -p modes. When -I or --create-index is also specified, use IF NOT EXISTS for index creation too. This option cannot be used with -d. Append mode requires an explicit creation action.

Atomic loader actions The short modes are presets over explicit actions. The equivalent long options are `--drop-table`, `--create-table`, `--load-data`, `--create-index`, `--add-constraints`, `--vacuum`, `--analyze`, and `--no-transaction`. Add `--if-not-exists` to make active creation actions use `IF NOT EXISTS`. `--drop-table` emits `DROP TABLE IF EXISTS` before the selected actions. With no mode specified, the default create/load actions still apply.

Options: `--help` `--help-all` `--help-section` `--help-sections`

- C, --add-constraints** `raster_columns` `table` `SRID`, `table`.
Add constraints to the raster column.
- x, --no-extent** Disable setting the max extent constraint. Only applied if `-C` or `--add-constraints` is also used.
- r, --regular-blocking** Set the constraints (spatially unique and coverage tile) for regular blocking. Only applied if `-C` or `--add-constraints` is also used.

Options: `--help` `--help-all` `--help-section` `--help-sections`

- s, --srid [`<FROM_SRID>:`]`<SRID>`** Assign the output raster to the specified SRID. When `FROM_SRID` is also provided, `raster2pgsql` marks the input raster with `FROM_SRID` and emits SQL that reprojects the raster to the target SRID. If the source SRID is omitted or is zero, the raster metadata will be checked to determine an appropriate source SRID. Reprojection cannot be used with copy mode.
- b, --band `BAND` (`1-255`) `table`**. `table`, `table` (`.`) `table`. `table`.
Specify the band number (1-255) for the raster. If not specified, the first band is used.
- t, --tile-size `TILE_SIZE` `table`**. `TILE_SIZE` `table` x `table`, `table` "auto" `table`.
Specify the tile size in pixels. If not specified, the default is 256x256.
- P, --pad `padding` (padding) `table`**.
Specify the padding value for the raster.
- R, --register `table` (DB `table`) `table`**.
Register the raster in the database (if `table`) and create the index.
- l, --overview-factor `OVERVIEW_FACTOR`** Create overview of the raster. For more than one factor, separate with comma(.). Overview table name follows the pattern `overview_factor_table` where `overview_factor` is a placeholder for numerical overview factor and `table` is replaced with the base table name. If a source raster already has an overview with matching dimensions, that overview is reused. Created overview is stored in the database and is not affected by `-R`. Note that your generated sql file will contain both the main table and overview tables.
- N, --nodata `NODATA` "NODATA" `table` `NODATA` `table`**.

Options: `--help` `--help-all` `--help-section` `--help-sections`

- f, --raster-column `COLUMN` `table`**. `table` 'rast' `table`.
- F, --filename `table`**.
- n, --filename-column `COLUMN` `table`**. `table` `table`.
- q, --quote** PostgreSQL `table`.
- I** Alias for `--create-index`.
- create-index** Create a GiST index on the raster column at the end of this `raster2pgsql` run. With repeated `-a` append runs, create the index on the final run or after loading; add `--if-not-exists` to make reruns tolerate an existing index.
- M `table` (vacuum analyze) `table`**.
- k, --skip-nodata-check** 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 tablespace** 指定表空间名称。-X 指定索引表空间名称。
- X, --index-tablespace tablespace** 指定索引表空间名称。-I 指定索引表空间名称。
- Y, --copy [MAX_ROWS_PER_COPY]** Use copy statements instead of insert statements. Optionally specify max_rows_per_copy; default 50 when not specified.
- e, --no-transaction** 禁用事务 (transaction)。
- E, --endian ENDIAN** 指定字节序 (endianness) 为 XDR 或 NDR。XDR 为 0, NDR 为 1。
- V, --wkb-version version** 指定 WKB 版本。0 或 1。

10.1.2 PostGIS 创建栅格表

以下示例将创建一个名为 myrasters 的表，用于存储栅格数据。该表包含一个主键列 rid 和一个栅格列 rast。

1. 创建表 myrasters:

```
CREATE TABLE myrasters(rid serial primary key, rast raster);
```

2. 向表中插入数据。首先，使用 `ST_MakeEmptyRaster` 函数创建一个空栅格，然后使用 `ST_AddBand` 函数添加波段。接着，使用 `ST_AsRaster` 函数将点数据转换为栅格，最后使用 `ST_Union` 和 `ST_MapAlgebra` 函数进行聚合运算 (algebra)。
3. 创建索引 myrasters_rast_st_convexhull_idx，使用 `gist(ST_ConvexHull(rast))` 作为索引函数。

```
CREATE INDEX myrasters_rast_st_convexhull_idx ON myrasters USING gist(ST_ConvexHull(rast));
```

凸包 (convex hull) 函数 `ST_ConvexHull` 用于计算栅格的凸包。



Note

PostGIS 2.0 引入了 `envelop` 函数，用于计算栅格的包络线。该函数在 PostGIS 2.0 中首次出现。

4. **AddRasterConstraints** 函数用于添加栅格约束。

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=xxxxxxxxxxxxxxxxxxxxxx \
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \
raster2pgsql \
-s 990000 \
-t 256x256 \
-I \
-R \
/vsis3/your.bucket.com/your_file.tif \
your_table \
| psql your_db
```

Once the table is loaded, you need to give the database permission to read from remote rasters, by setting two permissions, `postgis.enable_outdb_rasters` and `postgis.gdal_enabled_drivers`.

```
SET postgis.enable_outdb_rasters = true;
SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

To make the changes sticky, set them directly on your database. You will need to re-connect to experience the new settings.

```
ALTER DATABASE your_db SET postgis.enable_outdb_rasters = true;
ALTER DATABASE your_db SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

For non-public rasters, you may have to provide access keys to read from the cloud rasters. The same keys you used to write the `raster2pgsql` call can be set for use inside the database, with the `postgis.gdal_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=xxxxxxxxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
```

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 地理数据管理

PostGIS 提供了许多用于管理地理数据的函数。这些函数可以分为三类：用于加载数据的函数、用于查询数据的函数以及用于管理数据的函数。本章将介绍一些常用的函数。

1. `raster_columns` 用于查看数据库中所有栅格表的元数据。
2. `raster_overviews` 用于查看数据库中所有栅格表的概图元数据。

10.2.1 查看元数据

`raster_columns` 用于查看数据库中所有栅格表的元数据。该函数返回一个表，其中包含每个栅格表的基本信息，如表名、列名、数据类型、SRID、格式、分辨率、宽度和高度。

ラスターストリング, 列名を指定する。 raster_columns 列名を指定する。

ラスターストリングを指定する -c 列名を指定する, raster_columns 列名を指定する。 AddRasterConstraints 列名を指定する。

- r_table_catalog ラスターストリング。 ラスターストリングを指定する。
- r_table_schema ラスターストリング。
- r_table_name ラスターストリング。
- r_raster_column ラスターストリング r_table_name ラスターストリング。 PostGIS ラスターストリング, ラスターストリング, ラスターストリング。
- srid ラスターストリング。 Section 4.5 ラスターストリング。
- scale_x ラスターストリング (列) 列名。 ラスターストリング scale_x 列名, scale_x ラスターストリング。 ラスターストリング ST_ScaleX ラスターストリング。
- scale_y ラスターストリング (列) 列名。 ラスターストリング scale_y 列名, scale_y ラスターストリング。 ラスターストリング ST_ScaleY ラスターストリング。
- blocksize_x ラスターストリング (列名) 列名。 ラスターストリング ST_Width ラスターストリング。
- blocksize_y ラスターストリング (列名) 列名。 ラスターストリング ST_Height ラスターストリング。
- same_alignment ラスターストリング。 ラスターストリング ST_SameAlignment ラスターストリング。
- regular_blocking ラスターストリング, ラスターストリング。
- num_bands ラスターストリング。 ラスターストリング ST_NumBands
- pixel_types ラスターストリング。 ラスターストリング pixel_types ラスターストリング ST_BandPixelType ラスターストリング。
- nodata_values ラスターストリング nodata_value ラスターストリング (double precision) 列 (数) 列名。 ラスターストリング。 ラスターストリング ST_BandNoDataValue ラスターストリング。
- out_db ラスターストリング。 ラスターストリング。
- extent ラスターストリング (extent) 列名。 ラスターストリング DropRasterConstraints ラスターストリング AddRasterConstraints ラスターストリング。
- spatial_index ラスターストリング。

10.2.2 ラスターストリング

raster_overviews ラスターストリング raster_columns ラスターストリング。 ラスターストリング, ラスターストリング, ラスターストリング。 ラスターストリング -l ラスターストリング。 AddOverviewConstraints ラスターストリング。

ラスターストリング。



Note

```
raster_overviews || raster_columns || raster_columns
|| raster_overviews || raster_columns ||
|| (join) ||
```

[illegible]

- [illegible]

```
raster_overviews ████████████████████████████████████████.
```

- `o_table_catalog` 数据库名称. 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` 数据库名称. 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. 栅格图像缩小的倍数. `raster2pgsql` 默认 1 倍. 2 倍. 4 倍. 125x125 5000x5000 栅格图像, $(5000*5000)/(125*125) = 1600$ 倍, `o_2` ($l=2$) $(1600/2^2) = 400$ 倍, `o_3` ($l=3$) $(1600/2^3) = 200$ 倍. (scrap) (栅格图像缩小的倍数) `raster2pgsql` 默认 2^overview factor 倍.

10.3 PostGIS

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 中生成 PNG

在 PHP 中，PHP PostgreSQL 扩展使用 **ST_AsGDALRaster** 函数（见 1, 2, 3 节）通过 PHP 请求流（request stream）生成 PNG 数据。在 PHP 中，使用 `img src` HTML 属性来显示 PNG 数据。

在 PHP 中，使用 `(combine)` 函数将 WGS84 坐标转换为 UTM 坐标，使用 **ST_Union** 函数（见 10.3.1 节）生成 union 数据，使用 **ST_Transform** 函数（见 10.3.1 节）将 union 数据转换为 UTM 坐标，使用 **ST_AsPNG** 函数生成 PNG 数据。

http://mywebserver/test_raster.php?srid=2249

在 PHP 中，使用 `(combine)` 函数将 WGS84 坐标转换为 UTM 坐标，使用 `(union)` 函数生成 union 数据，使用 `(ST_Transform)` 函数将 union 数据转换为 UTM 坐标，使用 `(ST_AsPNG)` 函数生成 PNG 数据。

```
<?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 ensures bytea values are returned using escape format. */
$sql = "set bytea_output='escape';
SELECT ST_AsPNG(
    ST_Transform(
        ST_AddBand(
            ST_Union(rast, 1),
            ARRAY[ST_Union(rast, 2), ST_Union(rast, 3)]
        ),
        $input_srid
    )
) As new_rast
FROM aerials.boston
WHERE
    ST_Intersects(rast, ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, -71.1210, ↔
        42.218,4326),26986) )";
$result = pg_query($sql);
$row = pg_fetch_row($result);
pg_free_result($result);
if ($row === false) return;
echo pg_unescape_bytea($row[0]);
?>
```

10.3.2 使用 ST_AsPNG 在 ASP.NET C# 中生成 PNG

在 PHP 中，npgsql PostgreSQL .NET 扩展使用 **ST_AsGDALRaster** 函数（见 1, 2, 3 节）通过 PHP 请求流（request stream）生成 PNG 数据。在 PHP 中，使用 `img src` HTML 属性来显示 PNG 数据。

在 PHP 中，使用 `npgsql PostgreSQL .NET` 扩展（见 <http://npgsql.projects.postgresql.org/>）生成 union 数据。在 ASP.NET bin 目录中，使用 `(ST_Transform)` 函数将 union 数据转换为 UTM 坐标，使用 `(ST_AsPNG)` 函数生成 PNG 数据。

在 PHP 中，使用 `(combine)` 函数将 WGS84 坐标转换为 UTM 坐标，使用 **ST_Union** 函数（见 10.3.1 节）生成 union 数据，使用 **ST_Transform** 函数（见 10.3.1 节）将 union 数据转换为 UTM 坐标，使用 **ST_AsPNG** 函数生成 PNG 数据。

下面 C# 代码在 [Section 10.3.1](#) 中进行了修改。

http://mywebserver/test_raster.php?srid=2249

下面代码在 [Section 10.3.1](#) 中进行了修改，以使用 [Section 10.3.1](#) 中的配置。

```
-- 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 aeriāls.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 Java 保存查询结果

本节将介绍如何使用 Java 保存查询结果。

<http://jdbc.postgresql.org/download.html> 下载 PostgreSQL JDBC 驱动程序。

编译和运行 SaveQueryImage.java 的步骤如下：

```

set env CLASSPATH ../postgresql-9.0-801.jdbc4.jar
javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class

```

编译和运行 SaveQueryImage.jar 的步骤如下：

```

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 简体中文 SQL 简体中文

PLPython 是 PostgreSQL 9.5 版本引入的。PLPython 是 PostgreSQL 9.5 版本引入的。PLPython 是 PostgreSQL 9.5 版本引入的。PLPython 是 PostgreSQL 9.5 版本引入的。

```

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, '8 ←
    BUI',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 导出数据到本地文件

使用 PSQL 导出数据到本地文件。 PostgreSQL 使用 `lo_export` 函数将大型对象导出到本地文件。 PSQL 使用 `\lo_export` 命令，将大型对象导出到本地文件。

使用以下命令，将大型对象导出到本地文件。

```
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);
```


Section 10.1

```
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, ('01000003009A99999999999A93F9A999999999A9BF000000E02B274A' ||
'410000000077195641000000000000000000000000000000000000 ←
FFFFFFFFFF050005000400FDFFDFEFDFEFDFEFDF9FAFEF' ||
' ←
EFCF9FBFDFEFDFCFAFEFEFE04004E627AADD16076B4F9FE6370A9F5FE59637AB0E54F58617087040046566487A1506
')::rast);
```

The functions below are ones which a user of PostGIS Raster is likely to need. There are other raster support functions which are not of interest to a general user.

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 1-1000。 NULL 返回, 返回。

pixeltype text **ST_BandPixelType** 返回。

initialvalue double precision 返回。

nodataval double precision 返回 NODATA 值。 NULL 返回, 返回 NODATA 值。

例

ST_AddBand

11.1.3 rastbandarg

rastbandarg — 返回 geometry 值。

例

返回 geometry 值。

rast raster 返回。

nband integer 返回 1-1000。

1. $[a-b] = a \leq x \leq b$
2. $(a-b) = a < x \leq b$
3. $[a-b) = a \leq x < b$
4. $(a-b) = a < x < b$

'(' 闭开区间 (a-b) 左 a-b 闭开区间。

pixeltype text ST_BandPixelType 数据类型。

nodataval double precision NODATA 数据类型。数据类型，数据类型。

值: 值 2 值 255 值 NODATA 值 8BUI 数据类型。

```
SELECT ROW(2, '0-100:1-10, 101-500:11-150,501 - 10000: 151-254', '8BUI', 255)::reclassarg;
```

值: 值 1 值 NODATA 数据类型 1BB 数据类型。

```
SELECT ROW(1, '0-100]:0, (100-255:1', '1BB', NULL)::reclassarg;
```

值

ST_Reclass

11.1.6 summarystats

summarystats — **ST_SummaryStats** 值 **ST_SummaryStatsAgg** 数据类型。

值

ST_SummaryStats 值 **ST_SummaryStatsAgg** 数据类型。

count integer 数据类型。

sum double precision 数据类型。

mean double precision 数据类型。

stddev double precision 数据类型。

min double precision 数据类型。

max double precision 数据类型。

值

ST_SummaryStats, ST_SummaryStatsAgg

11.1.7 unionarg

unionarg — 对 UNION 操作的结果应用 ST_Union 函数。

语法

ST_Union (UNION 操作的结果) ST_Union 函数。

nband integer 指定要使用的波段数。1-255。

uniontype text UNION 操作的结果。 **ST_Union** 函数。

语法

ST_Union

11.2 栅格函数

11.2.1 AddRasterConstraints

AddRasterConstraints — 为加载的栅格表添加特定列的约束，包括空间参考、缩放、块大小、对齐、波段、波段类型以及标志，用于指示栅格列是否定期分块。表必须加载数据以推断约束。返回 true 如果约束设置成功，否则返回 false。

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[] VARIADIC constraints);

boolean **AddRasterConstraints**(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=false, boolean num_bands=true, boolean pixel_types=true, boolean no_data_values=true, boolean out_db=true, boolean extent=true);

语法

raster_columns 表名, 列名, srid, SPATIAL_REF_SYS 名称, 块大小, 缩放, 对齐, 波段, 波段类型, 无数据值, 输出数据库, 范围。

raster2pgsql 表名, 列名, srid, SPATIAL_REF_SYS 名称, 块大小, 缩放, 对齐, 波段, 波段类型, 无数据值, 输出数据库, 范围。

Section 10.2.1 栅格函数。

- **blocksize** X Y 块大小。

- `blocksize_x` X 大小 (像素/块) 的块大小。
- `blocksize_y` Y 大小 (像素/块) 的块大小。
- `extent` 指定栅格在 X 和 Y 方向上的范围。指定为 `xmin, xmax, ymin, ymax`。
- `num_bands` 波段数。
- `pixel_types` 指定每个波段的像素数据类型。格式为 `N 数据类型`，其中 `N` 是波段数。
- `regular_blocking` 指定是否使用规则分块 (所有块大小相同)。默认为 `TRUE`。
- `same_alignment` 确保它们都具有相同的对齐方式，意味着任何两个你比较的栅格都会返回 `true`。参考 [ST_SameAlignment](#)。
- `srid` 指定 SRID。
- `--` 指定其他选项。



Note

如果指定了 `extent`，则 `blocksize_x` 和 `blocksize_y` 必须为正数。



Note

如果指定了 `num_bands`，则 `pixel_types` 必须指定与 `num_bands` 相同的数量。使用 `DropRasterConstraints` 可以删除约束。

2.0.0 版本更新。

创建表并插入数据：

```
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: 创建表并添加栅格

```
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');
```

NOTICE: Adding regular blocking constraint
 NOTICE: Adding blocksize-X constraint
 NOTICE: Adding blocksize-Y constraint

例

Section [10.2.1, ST_AddBand](#), [ST_MakeEmptyRaster](#), [DropRasterConstraints](#), [ST_BandPixelType](#), [ST_SRID](#)

11.2.2 DropRasterConstraints

DropRasterConstraints — 删除栅格表的约束。PostGIS 3.7.0alpha1 新增。在 3.7.0alpha1 之前，该函数不存在。

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);

例

AddRasterConstraints 例, 删除栅格表的约束。PostGIS 3.7.0alpha1 新增。在 3.7.0alpha1 之前，该函数不存在。

例: 删除栅格表的约束。

```
DROP TABLE mytable
```

例: 删除栅格表的约束, 使用 SQL 语句。

```
ALTER TABLE mytable DROP COLUMN rast
```

例: 删除栅格表的约束 raster_columns 表。在 3.7.0alpha1 之前，该函数不存在。

2.0.0 版本。

例

```
SELECT DropRasterConstraints ('myrasters','rast');
----RESULT output ---
t

-- verify change in raster_columns --
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types, ↵
       nodata_values
FROM raster_columns
WHERE r_table_name = 'myrasters';
```

srid	scale_x	scale_y	blocksize_x	blocksize_y	num_bands	pixel_types	↵
nodata_values							
0							

例

AddRasterConstraints

11.2.3 AddOverviewConstraints

AddOverviewConstraints — 表 (overview) 。

Synopsis

boolean **AddOverviewConstraints**(name ovschema, name ovtable, name ovcolumn, name refschema, name reftable, name refcolumn, int ovfactor);
boolean **AddOverviewConstraints**(name ovtable, name ovcolumn, name reftable, name refcolumn, int ovfactor);

例

raster_overviews 。

ovfactor (乗数) 。

ovschema refschema, search_path 。

2.0.0 。

例

```
CREATE TABLE res1 AS SELECT
ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0, 0, 2),
1, '8BSI'::text, -129, NULL
) r1;

CREATE TABLE res2 AS SELECT
ST_AddBand(ST_MakeEmptyRaster(500, 500, 0, 0, 4),
```


例

PostGIS 通过 GDAL 支持栅格数据。GDAL 支持栅格数据格式。

例

```
SELECT PostGIS_GDAL_Version();
```

```
GDAL 1.11dev, released 2013/04/13
```

例

[postgis.gdal_datapath](#)

11.2.6 PostGIS_Raster_Lib_Build_Date

PostGIS_Raster_Lib_Build_Date — 返回栅格库的构建日期。

Synopsis

text **PostGIS_Raster_Lib_Build_Date()**;

例

返回栅格库的构建日期。

例

```
SELECT PostGIS_Raster_Lib_Build_Date();
```

```
2010-04-28 21:15:10
```

例

[PostGIS_Raster_Lib_Version](#)

11.2.7 PostGIS_Raster_Lib_Version

PostGIS_Raster_Lib_Version — 返回栅格库的版本。

Synopsis

text **PostGIS_Raster_Lib_Version()**;

例

PostGIS_Raster_Lib_Version().

例

```
SELECT PostGIS_Raster_Lib_Version();
```

```
2.0.0
```

例

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);

例

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.

例: 2.0.6, 2.1.3 例 - GUC `gdal_enabled_drivers` 例, 例

2.0.0 例. GDAL 1.6.0 例.

例: 例

```
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 (.sdatt, .sg-grd-z)	t
SAR_CE0S	CE0S 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

例: 输出JPEG的创建选项XML列

Output the create options XML column of JPEG as a table. These creator options can be used in the 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)

This example returns raw XML output for GeoTIFF creator options.

```
SELECT create_options
FROM ST_GDALDrivers()
WHERE short_name = 'GTiff';

<CreationOptionList>
  <Option name="COMPRESS" type="string-select">
    <Value>
>NONE</Value>
    <Value>
>LZW</Value>
    <Value>
>PACKBITS</Value>
    <Value>
>JPEG</Value>
    <Value>
>CCITTRLE</Value>
    <Value>
>CCITTFAX3</Value>
    <Value>
>CCITTFAX4</Value>
    <Value>
>DEFLATE</Value>
  </Option>
  <Option name="PREDICTOR" type="int" description="Predictor Type"/>
  <Option name="JPEG_QUALITY" type="int" description="JPEG quality 1-100" default="75"/>
  <Option name="ZLEVEL" type="int" description="DEFLATE compression level 1-9" default ↵
    ="6"/>
  <Option name="NBITS" type="int" description="BITS for sub-byte files (1-7), sub-uint16 ↵
    (9-15), sub-uint32 (17-31)"/>
  <Option name="INTERLEAVE" type="string-select" default="PIXEL">
    <Value>
>BAND</Value>
    <Value>
```

```

>PIXEL</Value>
  </Option>
  <Option name="TILED" type="boolean" description="Switch to tiled format"/>
  <Option name="TFW" type="boolean" description="Write out world file"/>
  <Option name="RPB" type="boolean" description="Write out .RPB (RPC) file"/>
  <Option name="BLOCKXSIZE" type="int" description="Tile Width"/>
  <Option name="BLOCKYSIZE" type="int" description="Tile/Strip Height"/>
  <Option name="PHOTOMETRIC" type="string-select">
    <Value
>MINISBLACK</Value>
    <Value
>MINISWHITE</Value>
    <Value
>PALETTE</Value>
    <Value
>RGB</Value>
    <Value
>CMYK</Value>
    <Value
>YCBCR</Value>
    <Value
>CIELAB</Value>
    <Value
>ICCLAB</Value>
    <Value
>ITULAB</Value>
  </Option>
  <Option name="SPARSE_OK" type="boolean" description="Can newly created files have ↵
    missing blocks?" default="FALSE"/>
  <Option name="ALPHA" type="boolean" description="Mark first extrasample as being alpha ↵
    "/>
  <Option name="PROFILE" type="string-select" default="GDALGeoTIFF">
    <Value
>GDALGeoTIFF</Value>
    <Value
>GeoTIFF</Value>
    <Value
>BASELINE</Value>
  </Option>
  <Option name="PIXELTYPE" type="string-select">
    <Value
>DEFAULT</Value>
    <Value
>SIGNEDBYTE</Value>
  </Option>
  <Option name="BIGTIFF" type="string-select" description="Force creation of BigTIFF file ↵
    ">
    <Value
>YES</Value>
    <Value
>NO</Value>
    <Value
>IF_NEEDED</Value>
    <Value
>IF_SAFER</Value>
  </Option>
  <Option name="ENDIANNESS" type="string-select" default="NATIVE" description="Force ↵
    endianness of created file. For DEBUG purpose mostly">
    <Value
>NATIVE</Value>
    <Value
>INVERTED</Value>

```

```

    <Value
>LITTLE</Value>
    <Value
>BIG</Value>
  </Option>
  <Option name="COPY_SRC_OVERVIEWS" type="boolean" default="NO" description="Force copy
of overviews of source dataset (CreateCopy())"/>
</CreationOptionList>

-- Output the create options XML column for GTiff as a table --
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip,
       array_to_string(xpath('Value/text()', g.opt),', ') As vals
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM ST_GDALDrivers()
WHERE short_name = 'GTiff') As g;

```

oname	otype	descrip	vals
COMPRESS	string-select		NONE, LZW,
PACKBITS, JPEG, CCITTRLE, CCITTFAX3, CCITTFAX4, DEFLATE			
PREDICTOR	int	Predictor Type	
JPEG_QUALITY	int	JPEG quality 1-100	
ZLEVEL	int	DEFLATE compression level 1-9	
NBITS	int	BITS for sub-byte files (1-7), sub-uint16 (9-15), sub-uint32 (17-31)	
INTERLEAVE	string-select		BAND, PIXEL
TILED	boolean	Switch to tiled format	
TFW	boolean	Write out world file	
RPB	boolean	Write out .RPB (RPC) file	
BLOCKXSIZE	int	Tile Width	
BLOCKYSIZE	int	Tile/Strip Height	
PHOTOMETRIC	string-select		MINISBLACK,
MINISWHITE, PALETTE, RGB, CMYK, YCBCR, CIELAB, ICCLAB, ITULAB			
SPARSE_OK	boolean	Can newly created files have missing blocks?	
ALPHA	boolean	Mark first extrasample as being alpha	
PROFILE	string-select		GDALGeoTIFF,
GeoTIFF, BASELINE			
PIXELTYPE	string-select		DEFAULT,
SIGNEDBYTE			
BIGTIFF	string-select	Force creation of BigTIFF file	YES, NO, IF_NEEDED, IF_SAFER
ENDIANNESS	string-select	Force endianness of created file. For DEBUG purpose	
mostly			NATIVE, INVERTED, LITTLE, BIG

COPY_SRC_OVERVIEWS	boolean	Force copy of overviews of source dataset (CreateCopy ↵
(())		
(19 rows)		

☐☐

`ST_AsGDALRaster`, `ST_SRID`, `postgis.gdal_enabled_drivers`

11.2.9 UpdateRasterSRID

`UpdateRasterSRID` — 将栅格数据集的 SRID 更改为指定值。

Synopsis

raster **UpdateRasterSRID**(name schema_name, name table_name, name column_name, integer new_srid);
raster **UpdateRasterSRID**(name table_name, name column_name, integer new_srid);

☐☐

将栅格数据集的 SRID 更改为指定值。如果指定了 schema_name，则 SRID 更改将应用于 schema_name 中的表 table_name 中的列 column_name。如果未指定 schema_name，则 SRID 更改将应用于当前 schema 中的表 table_name 中的列 column_name (☐☐, 将 SRID 更改为指定值)。



Note

如果指定了 schema_name，则 SRID 更改将应用于 schema_name 中的表 table_name 中的列 column_name。如果未指定 schema_name，则 SRID 更改将应用于当前 schema 中的表 table_name 中的列 column_name (☐☐, 将 SRID 更改为指定值)。

2.1.0 版本中引入。

☐☐

`UpdateGeometrySRID`

11.2.10 ST_CreateOverview

`ST_CreateOverview` — 为栅格数据集创建概述。

Synopsis

regclass **ST_CreateOverview**(regclass tab, name col, int factor, text algo='NearestNeighbor');

upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	↔
numbands									
0	0	100	100	1	-1	0	0	0	↩
2									

测试: 测试测试测试测试

```
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	outdbbandnum	filesize	↔
filetimestamp							
1	8BUI	0	f	null	null	null	↩
null							
2	32BF	-1	f	null	null	null	↩
null							
3	32BUI	12	f	null	null	null	↩
null							
4	16BUI	2	f	null	null	null	↩
null							

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 that The ORDER BY test_type clause relies on aggregate ORDER BY support. 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;
```

测试: 测试 DB 测试测试

```
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 简体中文 PostGIS 简体中文。

Synopsis

raster **ST_AsRaster**(geometry geom, raster ref, text pixeltype, double precision value=1, double precision nodataval=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, raster ref, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], boolean touched=false);

raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, integer width, integer height, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, integer width, integer height, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, integer width, integer height, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);



PostGIS 简体中文

This example maps bands to RGB channels, producing the teal value (118,154,118).

```
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]));
```

简体中文

[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_AsRasterAgg**(geometry geom, double precision val, raster ref, text pixeltype, double precision nodataval, text uniontype, boolean touched);

简体中文

Returns a single-band raster containing the rendered version of all incoming geometries, each with its associated value.

Availability: 3.6.0

简体中文

```
WITH inp(g,v) AS (
    VALUES
        ( ST_Buffer(ST_MakePoint(10,0), 10), 1 ),
        ( ST_Buffer(ST_MakePoint(20,0), 10), 2 )
),
agg AS (
    SELECT ST_AsRasterAgg(
        g,
        v,
```

```

        ST_MakeEmptyRaster(0,0,0,0,1.0),
        '8BUI',
        99,
        'SUM',
        true
    ) r
FROM inp
)
SELECT
    ST_Width(r) w,
    ST_Height(r) h,
    ST_Value(r,'POINT(5 0)') v5_0,
    ST_Value(r,'POINT(15 0)') v15_0,
    ST_Value(r,'POINT(25 0)') v25_0
FROM agg;

```

```

w | h | v5_0 | v15_0 | v25_0
---+---+-----+-----+-----
30 | 20 | 1 | 3 | 2
(1 row)

```

☐☐

[ST_AsRaster](#), [ST_DumpAsPolygons](#), [ST_Union](#)

11.3.4 ST_Band

ST_Band — 从现有栅格中提取指定波段并返回新栅格。如果未指定波段或指定的波段不存在，则返回所有波段。

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



如果 nbands 包含空字符串，则会导致错误。例如，`ST_Band(rast, '1,2,3')` 会返回错误。应使用 `ST_Band(rast, '1@2@3', '@')` 或 `ST_Band(rast, '{1,2,3}'::int[])` 来指定波段。PostGIS 3.7.0 之前，`ST_Band(rast, '1,2,3')` 会返回所有波段。

2.0.0 版本中，`ST_Band` 函数已弃用。

例

This example creates two rasters: one containing band 1 of dummy_rast, and one containing band 2 reclassified as 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.

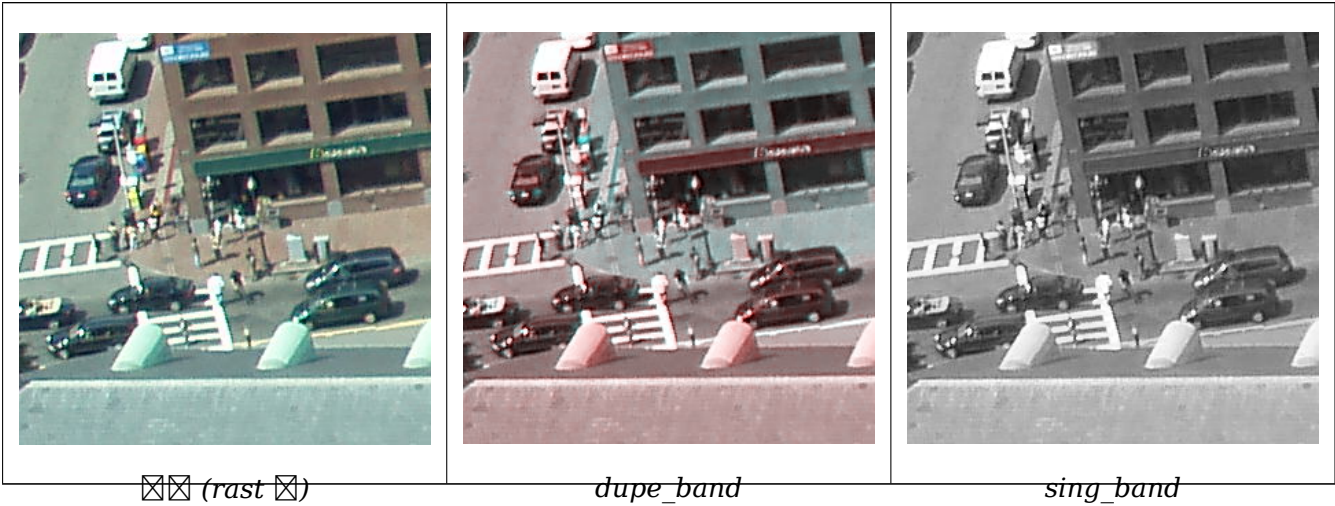
```
SELECT ST_NumBands(ST_Band(rast, '{2,3}':int[])) As num_bands
FROM dummy_rast WHERE rid=2;
```

2

Return bands 2 and 3 using array syntax.

```
SELECT ST_NumBands(ST_Band(rast, ARRAY[2,3])) As num_bands
FROM dummy_rast
WHERE rid=2;
```

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;
```


❏

```
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;
```

rid	upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	
3	0.0005	0.0005	100	100	1	1	0	0	4326	↩
4	0.0005	0.0005	100	100	1	1	0	0	4326	↩

❏

ST_AddBand, ST_MetaData, ST_ScaleX, ST_ScaleY, ST_SetValue, ST_SkewX, , ST_SkewY

11.3.7 ST_Tile

ST_Tile — 将栅格数据分割成指定大小的瓦片。

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);

❏

将栅格数据分割成指定大小的瓦片。

padwithnodata = FALSE 时，如果栅格数据在指定宽度和高度范围内没有数据，则返回 NULL。padwithnodata = TRUE 时，如果栅格数据在指定宽度和高度范围内没有数据，则返回 NODATA (padding) 值。NODATA (padding) 值由 nodataval 参数指定。如果 nodataval 参数未指定，则返回 NODATA 值。

**Note**

PostGIS 3.7.0alpha1 版本支持 DB 12.1，PostGIS 3.7.0alpha1 版本支持 DB 12.1。

2.1.0 版本更新。

更新

```
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;
```

```
(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, ARRAY[2], 3, 3) AS rast FROM bar
)
SELECT
  ST_DumpValues(rast)
FROM baz;

```

```

(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 — 将栅格数据重新分块，将每个分块的大小指定为 tw 和 th。

Synopsis

SETOF raster **ST_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

图

`(sfx, sfy)` 图元 (`tw, th`) 图元, 图元 (`tab, col`) 图元 (`ext`) 图元。

图元 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', 图元 'Lanczos' 图元。图元 [GDAL Warp resampling methods](#) 图元。

2.2.0 图元。

图

[ST_CreateOverview](#)

11.3.9 ST_FromGDALRaster

`ST_FromGDALRaster` — 图元 GDAL 图元。

Synopsis

raster **ST_FromGDALRaster**(bytea gdaldata, integer srid=NULL);

图

图元 GDAL 图元。 `gdaldata` 图元 GDAL 图元。

`srid` 图元 NULL 图元, 图元 GDAL 图元 SRID 图元。 `srid` 图元, 图元 SRID 图元。

2.1.0 图元。

图

```
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)

图例

ST_AsGDALRaster

11.4 地理函数 (accessor)

11.4.1 ST_GeoReference

ST_GeoReference — 返回 (world) 坐标系的 GDAL 或 ESRI 格式。返回 GDAL 格式。

Synopsis

text **ST_GeoReference**(raster rast, text format=GDAL);

图例

图例 返回 (carriage) 坐标系的 GDAL 或 ESRI 格式。返回 GDAL 格式。返回 'GDAL' 或 'ESRI' 格式。

返回格式:

GDAL:

```
scalex
skewy
skewx
scaley
upperleftx
upperlefty
```

ESRI:

```
scalex
skewy
skewx
scaley
upperleftx + scalex*0.5
upperlefty + scaley*0.5
```

例

```
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

例

ST_SetGeoReference, ST_ScaleX, ST_ScaleY

11.4.2 ST_Height

ST_Height — 返回栅格的垂直高度。

Synopsis

integer **ST_Height**(raster rast);

例

返回栅格的高度。

例

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;
```

rid	rastheight
1	20
2	5

例

ST_Width

11.4.3 ST_IsEmpty

ST_IsEmpty — 如果栅格的宽度 (width = 0) 或高度 (height = 0) 为零，则返回 true。否则返回 false。

Synopsis

boolean **ST_IsEmpty**(raster rast);

返回

ST_IsEmpty (width = 0, height = 0) 返回 true。ST_IsEmpty (width = 1, height = 1) 返回 false。
2.0.0 版本中，ST_IsEmpty 返回 true。

返回

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0, 0))
```

```
f
```

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0, 0))
```

```
t
```

返回

ST_HasNoBand

11.4.4 ST_MemSize

ST_MemSize — 返回 raster 的内存大小 (字节)。

Synopsis

integer **ST_MemSize**(raster rast);

返回

ST_MemSize (rast) 返回 raster 的内存大小。

PostgreSQL 函数 pg_column_size, pg_size_pretty, pg_relation_size, pg_total_relation_size 返回 raster 的内存大小。

Note



pg_relation_size 返回 ST_MemSize 返回的内存大小。pg_relation_size 返回 TOAST 数据的大小。pg_column_size 返回 raster 的内存大小。pg_total_relation_size 返回 raster 的内存大小，TOAST 数据的大小。

2.2.0 版本中，ST_MemSize 返回 true。

例

```
SELECT ST_MemSize(ST_AsRaster(ST_Buffer(ST_Point(1,5),10,1000),150, 150, '8BUI')) AS rast_mem;

22568
```

例

11.4.5 ST_MetaData

ST_MetaData — 返回栅格元数据，包括 (skew), 偏斜, 栅格大小, 坐标系, 波段数。

Synopsis

record **ST_MetaData**(raster rast);

例

返回栅格元数据，包括 (skew), 偏斜, 栅格大小, 坐标系, 波段数。返回的元数据包括: upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands

例

```
SELECT rid, (foo.md).*
FROM (SELECT rid, ST_MetaData(rast) As md
FROM dummy_rast) As foo;
```

rid	upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	
1	0.5	0.5	10	20	2	3	0	0	0	
2	3427927.75	5793244	5	5	0.05	-0.05	0	0	0	

例

ST_BandMetaData, ST_NumBands

11.4.6 ST_NumBands

ST_NumBands — 返回栅格波段数。

Synopsis

integer **ST_NumBands**(raster rast);

[illegible]

```

rid | numbands
----+-----
  1 |      0
  2 |      3

```

11.4.7 ST_PixelHeight

Synopsis

rastheight	pixheight	scalex	scaley	skewx	skewy
20	3	2	3	0	0
5	0.05	0.05	-0.05	0	0

例: 0 返回栅格高度和宽度

```
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

例

[ST_PixelWidth](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SkewX](#), [ST_SkewY](#)

11.4.8 ST_PixelWidth

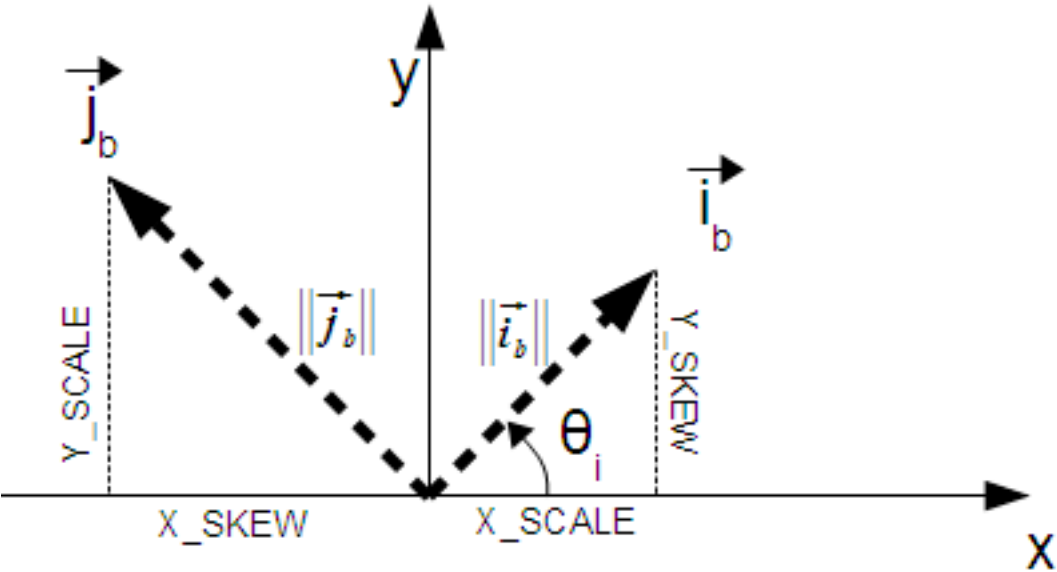
`ST_PixelWidth` — 返回栅格的像素宽度。

Synopsis

double precision **ST_PixelWidth**(raster rast);

例

返回栅格的像素宽度。如果栅格是倾斜的，则返回的像素宽度是倾斜后的像素宽度。
如果栅格是倾斜的，则返回的像素宽度是倾斜后的像素宽度。
如果栅格是倾斜的，则返回的像素宽度是倾斜后的像素宽度。



例: i 返回栅格高度
例: j 返回栅格宽度

图例: 栅格属性

```
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

图例: 0 度倾斜栅格

```
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

图例

ST_PixelHeight, ST_ScaleX, ST_ScaleY, ST_SkewX, ST_SkewY

11.4.9 ST_ScaleX

ST_ScaleX — 返回栅格的 X 轴缩放比例。

Synopsis

```
float8 ST_ScaleX(raster rast);
```

图例

返回栅格的 X 轴缩放比例。返回值为浮点型。图例: 2.0.0 度 WKTRaster 栅格的 ST_PixelSizeX 属性。

图例

```
SELECT rid, ST_ScaleX(rast) As rastpixwidth
FROM dummy_rast;
```

rid	rastpixwidth
1	2
2	0.05

返回

ST_Width

11.4.10 ST_ScaleY

ST_ScaleY — 将栅格的 Y 像素高度缩放为指定的值。

Synopsis

float8 **ST_ScaleY**(raster rast);

返回

返回栅格的 Y 像素高度。如果指定了第二个参数，则将高度缩放为指定的值。返回值的单位与输入栅格的单位相同。

版本: 2.0.0 在 WKTRaster 扩展中由 ST_PixelSizeY 函数实现。

返回

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;
```

rid	rastpixheight
1	3
2	-0.05

返回

ST_Height

11.4.11 ST_RasterToWorldCoord

ST_RasterToWorldCoord — 将栅格中的 X, Y 坐标转换为世界坐标。返回的 X, Y 坐标与栅格的单位相同。如果指定了第三个参数，则将 X 坐标缩放为指定的值。

Synopsis

record **ST_RasterToWorldCoord**(raster rast, integer xcolumn, integer yrow);

返回

返回栅格中的 X, Y 坐标。如果指定了第三个参数，则将 X 坐标缩放为指定的值。返回的 X, Y 坐标与栅格的单位相同。如果指定了第四个参数，则将 Y 坐标缩放为指定的值。返回的 X, Y 坐标与栅格的单位相同。

2.1.0 在 WKTRaster 扩展中实现。



Note

ST_ScaleX, ST_SkewX, and ST_RasterToWorldCoordX are deprecated. Use ST_ScaleX, ST_SkewX, and ST_RasterToWorldCoordX instead. ST_ScaleX, ST_SkewX, and ST_RasterToWorldCoordX are deprecated. Use ST_ScaleX, ST_SkewX, and ST_RasterToWorldCoordX instead.

Example: 2.1.0 ST_RasterToWorldCoordX

Example

This example shows that providing a column is sufficient for a non-skewed raster.

```
SELECT rid, ST_RasterToWorldCoordX(rast,1) As x1coord,
       ST_RasterToWorldCoordX(rast,2) As x2coord,
       ST_ScaleX(rast) As pixelx
FROM dummy_rast;
```

rid	x1coord	x2coord	pixelx
1	0.5	2.5	2
2	3427927.75	3427927.8	0.05

For fun let's skew it.

```
SELECT rid, ST_RasterToWorldCoordX(rast, 1, 1) As x1coord,
       ST_RasterToWorldCoordX(rast, 2, 3) As x2coord,
       ST_ScaleX(rast) As pixelx
FROM (SELECT rid, ST_SetSkew(rast, 100.5, 0) As rast FROM dummy_rast) As foo;
```

rid	x1coord	x2coord	pixelx
1	0.5	203.5	2
2	3427927.75	3428128.8	0.05

Example

ST_ScaleX, ST_RasterToWorldCoordY, ST_SetSkew, ST_SkewX

11.4.13 ST_RasterToWorldCoordY

ST_RasterToWorldCoordY — Returns the Y coordinate of the pixel at the given row. Returns 1 if the row is out of range.

Synopsis

```
float8 ST_RasterToWorldCoordY(raster rast, integer yrow);
float8 ST_RasterToWorldCoordY(raster rast, integer xcolumn, integer yrow);
```


图

图显示了栅格旋转后的结果。栅格旋转后，NaN 值保持不变。图显示了栅格旋转后的结果。

图

```
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

图

[ST_SetRotation](#), [ST_SetScale](#), [ST_SetSkew](#)

11.4.15 ST_SkewX

ST_SkewX — 对栅格 X 轴 (skew)(栅格) 进行倾斜。

Synopsis

```
float8 ST_SkewX(raster rast);
```

图

图显示了栅格 X 轴 (skew)(栅格) 进行倾斜后的结果。图显示了栅格 X 轴 (skew)(栅格) 进行倾斜后的结果。

图

```
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

图例

[ST_GeoReference](#), [ST_SkewY](#), [ST_SetSkew](#)

11.4.16 ST_SkewY

ST_SkewY — 对栅格 Y 轴进行 (逆时针/顺时针) 倾斜。

Synopsis

float8 **ST_SkewY**(raster rast);

图例

对栅格 Y 轴进行 (逆时针/顺时针) 倾斜。 倾斜角度 为 45 度。

图例

```
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

图例

[ST_GeoReference](#), [ST_SkewX](#), [ST_SetSkew](#)

11.4.17 ST_SRID

ST_SRID — spatial_ref_sys 表中的 SRID，返回栅格的 SRID。

Synopsis

integer **ST_SRID**(raster rast);

☐☐

spatial_ref_sys 表, 包含所有已知的空间参考系统。



Note

Non-georeferenced rasters and geometries use SRID 0.

☐☐

```
SELECT ST_SRID(rast) As srid
FROM dummy_rast WHERE rid=1;
```

```
0
```

☐☐

Section [4.5](#), [ST_SRID](#)

11.4.18 ST_Summary

ST_Summary — 返回栅格的摘要信息。

Synopsis

text **ST_Summary**(raster rast);

☐☐

返回的摘要信息如下。

2.1.0 版本开始支持。

☐☐

```
SELECT ST_Summary(ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
    1, '8BUI', 1, 0
),
    2, '32BF', 0, -9999
),
    3, '16BSI', 0, NULL
);
```

```
Raster of 10x10 pixels has 3 bands and extent of BOX(0 -10,10 0)+
band 1 of pixtype 8BUI is in-db with NODATA value of 0      +
band 2 of pixtype 32BF is in-db with NODATA value of -9999  +
band 3 of pixtype 16BSI is in-db with no NODATA value
(1 row)
```

返回

[ST_MetaData](#), [ST_BandMetaData](#), [ST_Summary](#) [ST_Extent](#)

11.4.19 ST_UpperLeftX

ST_UpperLeftX — 返回栅格左上角 X 坐标。

Synopsis

float8 **ST_UpperLeftX**(raster rast);

返回

返回栅格左上角 X 坐标。

返回

```
SELECT rid, ST_UpperLeftX(rast) As ulx
FROM dummy_rast;
```

rid	ulx
1	0.5
2	3427927.75

返回

[ST_UpperLeftY](#), [ST_GeoReference](#), [Box3D](#)

11.4.20 ST_UpperLeftY

ST_UpperLeftY — 返回栅格左上角 Y 坐标。

Synopsis

float8 **ST_UpperLeftY**(raster rast);

返回

返回栅格左上角 Y 坐标。

例

```
SELECT rid, ST_UpperLeftY(rast) As uly
FROM dummy_rast;
```

rid	uly
1	0.5
2	5793244

例

[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);

例

返回栅格的宽度。

例

```
SELECT ST_Width(rast) As rastwidth
FROM dummy_rast WHERE rid=1;
```

10

例

[ST_Height](#)

11.4.22 ST_WorldToRasterCoord

`ST_WorldToRasterCoord` — 将世界坐标 X, Y(经度, 纬度) 转换为栅格的行列坐标。

Synopsis

record **ST_WorldToRasterCoord**(raster rast, geometry pt);
record **ST_WorldToRasterCoord**(raster rast, double precision longitude, double precision latitude);

图 11.4.23

图 11.4.23 X, Y (xw, yw) 坐标转换。图 11.4.23 X, Y 坐标转换。图 11.4.23 X, Y 坐标转换。

2.1.0 版本中，ST_WorldToRasterCoordX 和 ST_WorldToRasterCoordY 函数。

图 11.4.24

```
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

图 11.4.25

ST_WorldToRasterCoordX, ST_WorldToRasterCoordY, ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

11.4.23 ST_WorldToRasterCoordX

ST_WorldToRasterCoordX — 将 (pt) 坐标转换为 (xw, yw) 坐标。

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);
```

图 11.4.26

图 11.4.26 (pt) 坐标转换为 (xw, yw) 坐标。图 11.4.26 (xw, yw) 坐标转换为 (pt) 坐标。

图 11.4.26: 2.1.0 版本中，ST_World2RasterCoordX 和 ST_World2RasterCoordY 函数。

图 11.4.27

```
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` — (pt) `X`, `Y` (xw, yw) `X`, `Y` `xw`, `yw`.

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);

(pt) `X`, `Y` (xw, yw) `X`, `Y`. (xw, yw) `xw` `yw`. `xw` `yw`.

2.1.0 `ST_World2RasterCoordY`.

```
SELECT rid, ST_WorldToRasterCoordY(rast,20.5) As ycoord,  
        ST_WorldToRasterCoordY(rast,3427927.8,20.5) As ycoord_xwyw,  
        ST_WorldToRasterCoordY(rast,ST_GeomFromText('POINT(3427927.8 20.5)',ST_SRID(rast))) As ptycoord  
FROM dummy_rast;
```

rid	ycoord	ycoord_xwyw	ptycoord
1	7	7	7
2	115864471	115864471	115864471

[ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SRID](#)

11.5 函数

11.5.1 ST_BandMetaData

ST_BandMetaData — 返回栅格带的元数据。返回 1 个记录。

Synopsis

- (1) record **ST_BandMetaData**(raster rast, integer band=1);
- (2) record **ST_BandMetaData**(raster rast, integer[] band);

返回

Returns basic meta data about a raster band. Columns returned: pixeltype, nodatavalue, isoutdb, path, outdbbandnum, filesize, filetimestamp.



Note
返回的元数据是只读的。



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.

示例: 返回 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	1	12345	1521807257	/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
3	8BUI		t	3	12345	1521807257	/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
2	8BUI		t	2	12345	1521807257	/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif

`ST_MetaData`, `ST_BandPixelType`

11.5.2 ST_BandNoDataValue

`ST_BandNoDataValue` — Returns the NODATA value for a given band. The NODATA value is the value that is used to represent missing data in a raster.

Synopsis

double precision **ST_BandNoDataValue**(raster rast, integer bandnum=1);

`ST_BandNoDataValue` returns the NODATA value for a given band.

```
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

ST_NumBands

11.5.3 ST_BandIsNoData

ST_BandIsNoData — NODATA

Synopsis

```
boolean ST_BandIsNoData(raster rast, integer band, boolean forceChecking=false);
```

```
boolean ST_BandIsNoData(raster rast, boolean forceChecking=false);
```



1. 如果 `isnodata` 为 `True`，则返回 `NODATA`。
 2. 如果 `isnodata` 为 `False`，则返回 `True`。

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.



Note

If the flag is dirty (this is, the result is different using TRUE as last parameter and not using it) you should update the raster to set this flag to true, by using `ST_SetBandIsNoData()`, or `ST_SetBandNoDataValue()` with TRUE as last argument. See [ST_SetBandIsNoData](#).



This example creates a 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 — . bandnum
 1 .

Synopsis

```
text ST_BandPath(raster rast, integer bandnum=1);
```


XXXXXXXXXXXXXXXXXXXXX. DB XXXXXXXXXXXXXXXXXXXXXXXX.



☒☒

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);

☒☒

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

☒☒

```
SELECT ST_BandFileSize(rast,1) FROM dummy_rast WHERE rid = 1;
```

```
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);

☒☒

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

☒☒

```
SELECT ST_BandFileTimestamp(rast,1) FROM dummy_rast WHERE rid = 1;
```

```
1521807257
```

11.5.7 ST_BandPixelType

ST_BandPixelType — 返回指定栅格 bandnum 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 返回的像素数据类型。可能的返回值如下：

- 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 字节的有符号整数
- 32BUI - 32 字节的无符号整数
- 32BF - 32 字节的浮点值
- 64BF - 64 字节的浮点值

返回

```
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
```

返回

ST_NumBands

11.5.8 ST_MinPossibleValue

ST_MinPossibleValue — 返回指定栅格的最小可能值。

Synopsis

integer **ST_MinPossibleValue**(text pixeltype);

例

返回栅格的最小可能值。

例

```
SELECT ST_MinPossibleValue('16BSI');
```

```
-32768
```

```
SELECT ST_MinPossibleValue('8BUI');
```

```
0
```

例

ST_BandPixelType

11.5.9 ST_HasNoBand

ST_HasNoBand — 返回栅格是否没有指定波段。返回 1 表示没有，0 表示有。

Synopsis

boolean **ST_HasNoBand**(raster rast, integer bandnum=1);

例

返回栅格是否没有指定波段。返回 1 表示没有，0 表示有。

2.0.0 版本引入。

例

```
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 栅格函数 (setter)

11.6.1 ST_PixelAsPolygon

ST_PixelAsPolygon — 将栅格像素转换为多边形。

Synopsis

geometry **ST_PixelAsPolygon**(raster rast, integer columnx, integer rowy);

¶

将栅格像素转换为多边形。

2.0.0 版本引入。

¶

This example returns a raster pixel polygon.

```
SELECT i,j, ST_AsText(ST_PixelAsPolygon(foo.rast, i,j)) As blpgeom
FROM dummy_rast As foo
  CROSS JOIN generate_series(1,2) As i
  CROSS JOIN generate_series(1,1) As j
WHERE rid=2;
```

i	j	blpgeom
1	1	POLYGON((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...
2	1	POLYGON((3427927.8 5793244,3427927.85 5793244,3427927.85 5793243.95, ..

¶

[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集合。X, Y 坐标。

Synopsis

setof record **ST_PixelAsPolygons**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

返回

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 返回多边形。ST_DumpAsPolygons 返回多边形。

2.0.0 返回记录格式。

返回: 2.1.0 返回 *exclude_nodata_value* 返回。

返回: 2.1.1 返回 *exclude_nodata_value* 返回。

返回

This example returns a 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
ORDER BY (gv).x, (gv).y;
```

x	y	val	geom
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))

返回

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsText

11.6.3 ST_PixelAsPoint

ST_PixelAsPoint — 返回记录格式。

Synopsis

geometry **ST_PixelAsPoint**(raster rast, integer columnx, integer rowy);

返回

返回栅格像素的几何点。

2.1.0 版本引入。

示例

```
SELECT ST_AsText(ST_PixelAsPoint(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;
```

```
POINT(0.5 0.5)
```

参见

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#)

11.6.4 ST_PixelAsPoints

ST_PixelAsPoints — 将栅格像素转换为点。返回 X, Y 坐标。

Synopsis

setof record **ST_PixelAsPoints**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

返回

返回栅格像素的几何点。返回 X, Y 坐标。返回栅格像素的几何点。

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 版本引入。

参数: 2.1.1 版本引入 *exclude_nodata_value* 参数。

SQL

```
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)

SQL

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#)

11.6.5 ST_PixelAsCentroid

ST_PixelAsCentroid — Returns the centroid of a pixel in a raster.

Synopsis

geometry **ST_PixelAsCentroid**(raster rast, integer x, integer y);

SQL

Returns the centroid of the pixel at the location (x, y) in the raster.

Availability: 2.1.0 or later C or PL/SQL.

2.1.0 or later PL/SQL.

例

```
SELECT ST_AsText(ST_PixelAsCentroid(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;
```

```
POINT(1.5 2)
```

例

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroids](#)

11.6.6 ST_PixelAsCentroids

`ST_PixelAsCentroids` — 将栅格中的每个像素 (像素值) 转换为 X, Y 坐标的几何点。返回记录格式: `geom geometry, val double precision, x integer, y integers`。

Synopsis

setof record **ST_PixelAsCentroids**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

例

将栅格中的每个像素 (像素值) 转换为 X, Y 坐标的几何点。返回记录格式: `geom geometry, val double precision, x integer, y integers`。

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.

- 例: 2.1.0 版本 C 语言实现。
- 例: 2.1.1 版本 `exclude_nodata_value` 参数实现。
- 2.1.0 版本实现。

例

This example uses LATERAL syntax.

```
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

ST Value — columnx, rowy [columnx, rowy]. If columnx or rowy are 1, exclude_nodata_value is ignored. If both are 1, nodata is ignored. exclude_nodata_value is ignored if both are 0.

```
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);
```

`exclude_nodata_value` 0x00000000, 0x00 nodata 0x0000000000000000. 0x00
0x00000000 exclude nodata value 0x0000000000.

The allowed values of the resample parameter are "nearest" which performs the default nearest-neighbor resampling, "bilinear" which performs a **bilinear interpolation** to estimate the value between pixel centers, and the nearest-neighbor boundary options "nearest-UL", "nearest-UR", "nearest-LL", and "nearest-LR". Boundary options choose which pixel is returned when the point lies on a horizontal or vertical pixel boundary, including corner intersections.

Enhanced: 3.7.0 resample accepts nearest-neighbor boundary options "nearest-UL", "nearest-UR", "nearest-LL", and "nearest-LR".

2.1.0 排除_nodata_value 排除_nodata_value。

2.0.0 排除_nodata_value 排除_nodata_value。

☐☐

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 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 5793243.85,3427927.9 5793243.85,3427927.85 5793243.85,3427927.85 5793243.9,3427927.9 5793243.9,3427927.9 5793243.95,
3427927.95 5793243.95,3427927.95 5793243.9),
((3427927.85 5793243.75,3427927.85 5793243.7,3427927.8 5793243.7,3427927.8 5793243.75,
3427927.8 5793243.8,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85 5793243.8,3427927.85 5793243.75)),
((3427928.05 5793243.75,3427928.05 5793243.7,3427928 5793243.7,3427927.95 5793243.7,3427927.95 5793243.75,3427927.95 5793243.8,3427928 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


```
                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 => '16BSI',
    initialvalue => 0,
    nodataval => -999
),
    1,
    1,
    1,
    newvalueset => ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8 ←
        ]]) AS rast
)
SELECT
ST_AsText(ST_SetZ(
    rast,
    band => 1,
    geom => 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)::geometry',
    resample => 'bilinear'
))
FROM test_raster

```

```
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.

例

`columnx` 和 `rowy`, 返回指定像素的 `distanceX` 和 `distanceY` 值。如果指定了 `NODATA` 值, 则返回 2。如果指定了 `distanceX` 和 `distanceY` 值, 则返回 3。如果指定了 `X` 和 `Y` 值, 则返回 2。如果指定了 `columnx` 和 `rowy` 值, 则返回 2。

如果指定了 `bandnum` 值, 则返回指定带的值。如果指定了 `exclude_nodata_value` 值, 则返回指定的值。如果指定了 `nodata` 值, 则返回指定的值。如果指定了 `exclude_nodata_value` 值, 则返回指定的值。



Note

返回 2 的值是 $2 * (\text{distanceX}|\text{distanceY}) + 1$ 。如果指定了 `distanceX` 和 `distanceY` 值, 则返回 1。如果指定了 `distanceX` 和 `distanceY` 值, 则返回 3x3 的像素。



Note

`ST_Min4ma`, `ST_Sum4ma`, `ST_Mean4ma` 返回指定像素的 4 个像素的平均值。如果指定了 `bandnum` 值, 则返回指定带的值。如果指定了 `exclude_nodata_value` 值, 则返回指定的值。

2.1.0 像素邻域函数

例

This example reads a value from pixel 2x2.

```
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
```

```
{{NULL,1,1},{1,1,1},{1,NULL,1}}
```

This example reads a NODATA value from pixel 2x3.

```
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
```

$$\{\{1, 1, 1\}, \{1, \text{NULL}, 1\}, \{1, 1, 1\}\}$$
$$\{\{1, 1, 0\}, \{0, 1, 1\}, \{1, 1, 1\}\}$$

ST_NearestValue, ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

11.6.12 ST SetValue

Synopsis

```
raster ST_SetValue(raster rast, integer bandnum, geometry geom, double precision newvalue);
raster ST_SetValue(raster rast, geometry geom, double precision newvalue);
raster ST_SetValue(raster rast, integer bandnum, integer columnx, integer rowy, double precision
newvalue);
raster ST_SetValue(raster rast, integer columnx, integer rowy, double precision newvalue);
```

返回

Returns modified raster resulting from setting the specified pixels' values to new value for the designated band given the raster's row and column or a geometry. If no band is specified, then band 1 is assumed.

2.1.0 版本: `ST_SetValue()` 返回修改后的栅格。 `ST_SetValues()` 返回修改后的栅格 (wrapper) 返回。

返回

This example uses geometry input.

```
SELECT (foo.geomval).val, ST_AsText(ST_Union((foo.geomval).geom))
FROM (SELECT ST_DumpAsPolygons(ST_SetValue(
    rast,
    1,
    ST_Point(3427927.75, 5793243.95),
    50)
    ) As geomval
FROM dummy_rast
where rid = 2) As foo
WHERE (foo.geomval).val < 250
GROUP BY (foo.geomval).val;
```

val	st_astext
50	POLYGON((3427927.75 5793244,3427927.75 5793243.95,3427927.8 579324 ...
249	POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 57932 ...

This example stores the changed raster.

```
UPDATE dummy_rast SET rast = ST_SetValue(rast,1, ST_Point(3427927.75, 5793243.95),100)
WHERE rid = 2 ;
```

返回

ST_Value, ST_DumpAsPolygons

11.6.13 ST_SetValues

`ST_SetValues` — 返回修改后的栅格。

Synopsis

```
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[] []
newvalueset, boolean[] [] noset=NULL, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[] []
newvalueset, double precision nosetvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, integer width, integer
height, double precision newvalue, boolean keepnodata=FALSE);
```


+ + +		
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	9
2	3	9
3	1	9
3	2	9
3	3	9

```
/*
The ST_SetValues() does the following...

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      => | 1 | 9 |
```

```

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, ←
        0),
        1, '8BUI', 1, 0
    ),
    1, 1, 1,
    ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
    ARRAY[[false], [true]]::boolean[][])
    ) AS poly
) foo
ORDER BY 1, 2;

```

x	y	val
1	1	9
1	2	1
1	3	9
2	1	9
2	2	
2	3	9
3	1	9
3	2	9
3	3	9

```

/*
The ST_SetValues() does the following...

+ - + - + - +      + - + - + - +
|   | 1 | 1 |      |   | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      =>  | 1 |   | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, ←
        1, -1, 0, 0, 0),
        1, '8BUI', 1, 0
    ),
    1, 1, 1, NULL
    ),
    1, 1, 1,
    ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
    ARRAY[[false], [true]]::boolean[][],
    TRUE

```



```
    )
  ) AS poly
) foo
ORDER BY 1, 2;
```

x	y	val
1	1	
1	2	1
1	3	9
2	1	9
2	2	
2	3	9
3	1	9
3	2	9
3	3	9

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[[]], 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)			

简体中文

```
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)			

简体中文

[ST_Value](#), [ST_SetValue](#), [ST_PixelAsPolygons](#)

11.6.14 ST_DumpValues

ST_DumpValues — 将栅格中的值拆分为记录。

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  
);
```

图

图 2 显示了使用 `(ST_MakeEmptyRaster, ST_AddBand)` 的 `nband` 为 `NULL` 的 `ST_AddBand` 的结果。

2.1.0 图。

图

```
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;
```

5

图

`ST_Value`, `ST_SetValue`, `ST_SetValues`

11.6.15 ST_PixelOfValue

`ST_PixelOfValue` — 返回 `columnx`, `rowy` 处的像素值。

Synopsis

setof record **ST_PixelOfValue**(raster rast , integer nband , double precision[] search , boolean exclude_nodata_value=true);
setof record **ST_PixelOfValue**(raster rast , double precision[] search , boolean exclude_nodata_value=true);
setof record **ST_PixelOfValue**(raster rast , integer nband , double precision search , boolean exclude_nodata_value=true);
setof record **ST_PixelOfValue**(raster rast , double precision search , boolean exclude_nodata_value=true);

columnx, rowy. , 1

2.1.0.

```
SELECT
  (pixels).*
FROM (
  SELECT
    ST_PixelOfValue(ST_SetValue(ST_SetValue(ST_SetValue(ST_SetValue(ST_SetValue(
      ST_AddBand(ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
        '8BUI'::text, 1, 0
    ),
      1, 1, 0
    ),
      2, 3, 0
    ),
      3, 5, 0
    ),
      4, 2, 0
    ),
      5, 4, 255
    ),
    1, ARRAY[1, 255]) AS pixels
) AS foo
```

val	x	y
1	1	2
1	1	3
1	1	4
1	1	5
1	2	1
1	2	2
1	2	4
1	2	5
1	3	1
1	3	2
1	3	3
1	3	4
1	4	1
1	4	3
1	4	4

```

1 | 4 | 5
1 | 5 | 1
1 | 5 | 2
1 | 5 | 3
255 | 5 | 4
1 | 5 | 5

```

11.7 地理参考

11.7.1 ST_SetGeoReference

`ST_SetGeoReference` — 将地理参考信息设置到 6 个参数中。使用 GDAL 或 ESRI 格式。GDAL 格式。

Synopsis

raster **ST_SetGeoReference**(raster rast, text georefcoords, text format=GDAL);
 raster **ST_SetGeoReference**(raster rast, double precision upperleftx, double precision upperlefty,
 double precision scalex, double precision scaley, double precision skewx, double precision skewy);

参数

georefcoords 6 个参数。'GDAL' 或 'ESRI' 格式。GDAL 格式。6 个参数。NULL 表示默认值。

ESRI 格式:

GDAL:

```
scalex skewy skewx scaley upperleftx upperlefty
```

ESRI:

```
scalex skewy skewx scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5
```



Note

在 DB 中，使用 `ST_SetGeoReference` 时，必须指定格式。否则将导致错误。

注意: 2.1.0 版本中 `ST_SetGeoReference(raster, double precision, ...)` 已弃用。

示例

```

WITH foo AS (
  SELECT ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0) AS rast
)
SELECT
  0 AS rid, (ST_MetaData(rast)).*
FROM foo
UNION ALL

```



```
SELECT
  1, (ST_MetaData(ST_SetGeoReference(rast, '10 0 0 -10 0.1 0.1', 'GDAL'))).*
FROM foo
UNION ALL
SELECT
  2, (ST_MetaData(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	upperleftx srid	upperleftx 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	0	0	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 — 将栅格旋转到指定的角度。

Synopsis

raster **ST_SetRotation**(raster rast, float8 rotation);

☒☒

将栅格旋转到指定的角度。返回的栅格具有与输入栅格相同的元数据。如果输入栅格具有地理参考，则返回的栅格也将具有地理参考。

☒☒

```
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;
```



```
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)

图例

ST_ScaleX, ST_ScaleY, Box3D

11.7.4 ST_SetSkew

ST_SetSkew — 对 X 和 Y 轴 (skew)(以度为单位) 进行倾斜。 返回倾斜后的 X 和 Y 轴坐标。

Synopsis

```
raster ST_SetSkew(raster rast, float8 skewxy);
raster ST_SetSkew(raster rast, float8 skewx, float8 skewy);
```

图例

对 X 和 Y 轴 (以度为单位) 进行倾斜。 返回倾斜后的 X 和 Y 轴坐标。 图例显示了 **ST_SetSkew** 的结果。

图例

This is the first example.

```
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

This second example sets both values to the 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

☐☐

[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);

☐☐

将 SRID 与 `spatial_ref_sys` 表中的 SRID 名称关联。



Note

此函数返回的栅格具有与输入栅格相同的 SRID。如果输入栅格的 SRID 为 0，则返回的栅格的 SRID 将默认为 4326。

☐☐

Section [4.5](#), [ST_SRID](#)

11.7.6 ST_SetUpperLeft

`ST_SetUpperLeft` — 设置栅格左上角像素的投影 X 和 Y 坐标。

Synopsis

raster **ST_SetUpperLeft**(raster rast, double precision x, double precision y);

The default is NearestNeighbor which is the fastest but results in the worst interpolation.
 maxerr 0.125 0.125 0.125.



Note

GDAL Warp resampling methods 0.125 0.125 0.125.

2.0.0 0.125 0.125 0.125. GDAL 1.6.1 0.125 0.125 0.125.

Enhanced: 3.4.0 max and min resampling options added

0.125

```
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

0.125

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 scalexy, text algorithm=NearestNeighbor, double
precision maxerr=0.125);
raster ST_Rescale(raster rast, double precision scalex, double precision scaley, text algorithm=NearestNeighbor,
double precision maxerr=0.125);
```



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.

[illegible]

maxerr is the threshold for transformation approximation by the resampling algorithm (in pixel units). A default of 0.125 is used if no maxerr is specified, which is the same value used in GDAL gdalwarp utility. If set to zero, no approximation takes place.



Note

GDAL Warp resampling methods



Note

ST_Rescale ██████████ ST_SetScale ████████.
ST_SetScale ██████████ (██████) ████████
██████. ST_Rescale ██████████ ██████████
██████. ST_SetScale ██████████.

2.0.0 000000000000. GDAL 1.6.1 000000000000.

Enhanced: 3.4.0 max and min resampling options added

SRID: 2.1.0 SRID XXXXXXXXXXXX.



0.001 0.0015

This example returns 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
```

0.001

This example returns the pixel size of the rescaled raster.

```
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
```

0.0015



ST Resize, ST Resample, ST SetScale, ST ScaleX, ST ScaleY, ST Transform

11.7.9 ST_Reskew

ST Reskew — $\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$ (Bilanzierung) $\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$. NearestNeighbor($\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$), Bilinear, Cubic, CubicSpline $\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$ Lanczos $\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$. $\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$ NearestNeighbor $\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$.

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);
```



最近邻 (NearestNeighbor) 插值方法。NearestNeighbor(最近邻), Bilinear, Cubic, CubicSpline 和 Lanczos 插值方法。最近邻插值方法最近邻插值。

skewx ⊠ skewy ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠.

[illegible]

maxerr 0.125



Note

GDAL Warp resampling methods



Note

ST_Reskew ██████████ ST_SetSkew ████████.
ST_SetSkew ██████████ (██████) ██████
██████. ST_Reskew ██████████ ██████████
██████. ST_SetSkew ██████████.

2.0.0 20200128. GDAL 1.6.1 20200128.

SRID: 2.1.0 SRIDXXXXXXXXXXXX.

0.0 0.0015

```
SELECT ST_Rotation(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
    , '8BUI'::text, 1, 0));

0

-- the reskewed raster raster rotation
SELECT ST_Rotation(ST_Reskw(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, ←
    0, 0, 4269), '8BUI'::text, 1, 0), 0.0015));

-0.982793723247329
```

ST Resample, ST Rescale, ST SetSkew, ST SetRotation, ST SkewX, ST SkewY, ST Transform

ST_SnapToGrid — Snap geometry to grid. NearestNeighbor(*geom*, *srid*), Bilinear, Cubic, CubicSpline, Lanczos, NearestNeighbor.

```
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex=DEFAULT 0, double precision scaley=DEFAULT 0);
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, double precision scalexy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

`gridx & gridy` (`scalex & scaley`) `NearestNeighbor`, `Bilinear`, `Cubic`, `CubicSpline`, `Lanczos`. `NearestNeighbor`.

[illegible]

You can optionally define the pixel size of the new grid with `scalex` and `scaley`.

[illegible]

maxerr 0.125

**Note**

GDAL Warp resampling methods 。

**Note**

ST_Resample 。

2.0.0 。 GDAL 1.6.1 。

： 2.1.0 SRID 。

。

。

This example returns the upper-left X coordinate of the original raster.

```
SELECT ST_UpperLeftX(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));
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));
-0.0008
```

。

ST_Resample, ST_Rescale, ST_UpperLeftX, ST_UpperLeftY

11.7.11 ST_Resize

ST_Resize — 。

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 al-
gorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_Resize(raster rast, text width, text height, text algorithm=NearestNeighbor, double preci-
sion maxerr=0.125);
```


11.7.12 ST_Transform

ST_Transform — 将栅格数据从一种坐标系转换到另一种坐标系。支持的算法有：NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos。默认使用 NearestNeighbor 算法。

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);
```

返回

该函数使用指定的算法（像素变形）将栅格数据从一种坐标系转换到另一种坐标系。默认使用 NearestNeighbor 算法，maxerror 默认为 0.125。

支持的算法有：'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', 'Lanczos'。更多算法请参考 [GDAL Warp resampling methods](#)。

ST_Transform 与 **ST_SetSRID** 类似。ST_Transform 将栅格数据从一种坐标系转换到另一种坐标系（指定 SRID），ST_SetSRID 仅指定 SRID。

该函数支持 3 种 alignto 模式。默认使用 'same' 模式。更多选项请参考 [ST_Transform](#) 文档。

Note



If you find your transformation support is not working right, you may need to set the environment variable PROJSO to the projection library your PostGIS build 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. 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.

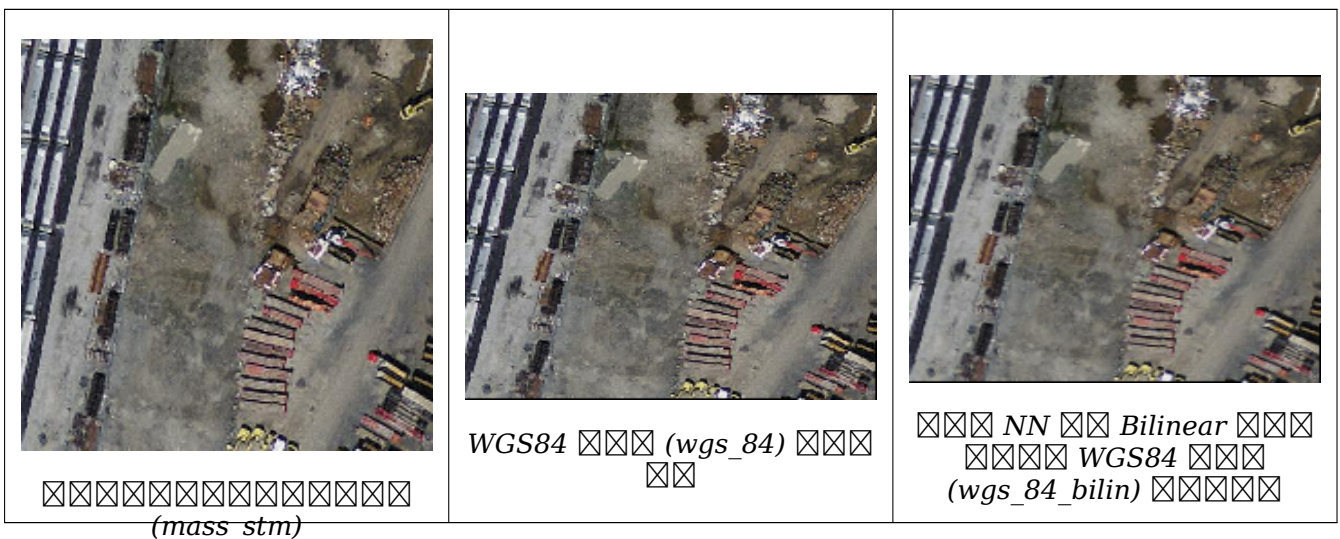
2.0.0 版本开始支持。GDAL 1.6.1 版本开始支持。

更新：2.1.0 版本开始支持 ST_Transform(rast, alignto) 函数。

返回

```
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



☒☒: ☒☒ 3

```
ST Transform(raster, srid) ST Transform(raster, alignto) ST Transform(raster, alignto, srid)
```

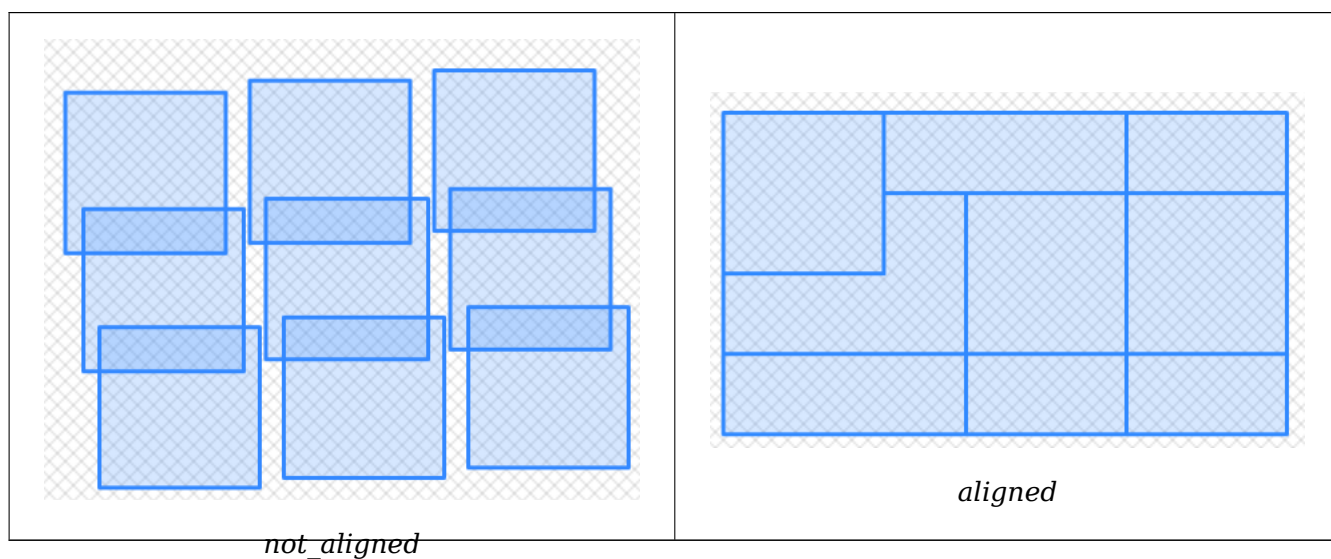
```
WITH foo AS (
  SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 600000, 100, -100, 0, 0, 2163), 1, '16BUI', 1, 0) AS rast UNION ALL
  SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 600000, 100, -100, 0, 0, 2163), 1, '16BUI', 2, 0) AS rast UNION ALL
  SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 600000, 100, -100, 0, 0, 2163), 1, '16BUI', 3, 0) AS rast UNION ALL

  SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599800, 100, -100, 0, 0, 2163), 1, '16BUI', 10, 0) AS rast UNION ALL
  SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599800, 100, -100, 0, 0, 2163), 1, '16BUI', 20, 0) AS rast UNION ALL
  SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599800, 100, -100, 0, 0, 2163), 1, '16BUI', 30, 0) AS rast UNION ALL

  SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599600, 100, -100, 0, 0, 2163), 1, '16BUI', 100, 0) AS rast UNION ALL
  SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599600, 100, -100, 0, 0, 2163), 1, '16BUI', 200, 0) AS rast UNION ALL
  SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599600, 100, -100, 0, 0, 2163), 1, '16BUI', 300, 0) AS rast
```

```
), bar AS (
    SELECT
        ST_Transform(rast, 4269) AS alignto
    FROM foo
    LIMIT 1
), baz AS (
    SELECT
        rid,
        rast,
        ST_Transform(rast, 4269) AS not_aligned,
        ST_Transform(rast, alignto) AS aligned
    FROM foo
    CROSS JOIN bar
)
SELECT
    ST_SameAlignment(rast) AS rast,
    ST_SameAlignment(not_aligned) AS not_aligned,
    ST_SameAlignment(aligned) AS aligned
FROM baz
```

rast	not_aligned	aligned
t	f	t



ST_Transform, ST_SetSRID

11.8 ☒☒☒☒☒☒☒☒

11.8.1 ST_SetBandNoDataValue

ST_SetBandNoDataValue — NODATA ¶¶¶¶¶¶¶¶¶¶¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶¶¶¶¶
 ¶ 1 ¶¶¶¶¶¶. ¶¶¶ NODATA ¶¶¶¶¶¶¶¶¶, no data value = NULL ¶¶¶¶¶¶¶¶.

Synopsis

raster **ST_SetBandNoDataValue**(raster rast, double precision nodatavalue);
 raster **ST_SetBandNoDataValue**(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);

返回

返回 NODATA 值。返回 1 个值。返回 **ST_Polygon**, **ST_DumpAsPolygons**, 返回 **ST_PixelAs...** 返回。

返回

This example changes only the first band NODATA 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;
```

返回

ST_BandNoDataValue, **ST_NumBands**

11.8.2 ST_SetBandIsNoData

ST_SetBandIsNoData — 返回 isnodata 值。

Synopsis

raster **ST_SetBandIsNoData**(raster rast, integer band=1);

返回

返回 isnodata 值。返回 1 个值。返回 **ST_BandIsNoData** 返回。

2.0.0 版本。

☒☒

This example creates a 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	pixeltype	nodatavalue	isoutdb	path	outdbbandnum	
1	1	8BUI		t	raster/test/regress/loader/Projected.tif	1	/home/pele/devel/geo/postgis-git/ ↩
1	2	8BUI		t	raster/test/regress/loader/Projected.tif	2	/home/pele/devel/geo/postgis-git/ ↩
1	3	8BUI		t	raster/test/regress/loader/Projected.tif	3	/home/pele/devel/geo/postgis-git/ ↩
2	1	8BUI		t	raster/test/regress/loader/Projected.tif	1	/home/pele/devel/geo/postgis-git/ ↩
2	2	8BUI		t	raster/test/regress/loader/Projected2.tif	1	/home/pele/devel/geo/postgis-git/ ↩
2	3	8BUI		t	raster/test/regress/loader/Projected.tif	3	/home/pele/devel/geo/postgis-git/ ↩



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	outdbbandnum	
1	1	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵	1	
1	2	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵	2	
1	3	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵	3	
2	1	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵	1	
2	2	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵	1	
2	3	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵	3	

❏❏

ST_BandMetaData, ST_SetBandPath

11.9

11.9.1 ST_Count

ST_Count — Count the number of non-null pixels in a raster. The count is returned as an integer. The count is 1 if the pixel is not null. exclude_nodata_value is a boolean that, if true, counts only non-null pixels. If false, all pixels are counted, including nulls.

Synopsis

bigint **ST_Count**(raster rast, integer nband=1, boolean exclude_nodata_value=true);
bigint **ST_Count**(raster rast, boolean exclude_nodata_value);

Count the number of pixels in the raster that are non-zero. The nband argument is optional and defaults to 1. The exclude_nodata_value argument is optional and defaults to true.



Note
exclude_nodata_value is true, the number of non-zero pixels is returned. If exclude_nodata_value is false, the number of non-zero pixels is returned, including the number of pixels with the value of the exclude_nodata_value.

2.2.0 **ST_Count**(rastertable, rastercolumn, ...) **ST_CountAgg**. **ST_CountAgg** **ST_Count**.

2.0.0 **ST_Count**.

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

ST_CountAgg, **ST_SummaryStats**, **ST_SetBandNoDataValue**

11.9.2 ST_CountAgg

ST_CountAgg — Count the number of pixels in the raster that are non-zero. The nband argument is optional and defaults to 1. The exclude_nodata_value argument is optional and defaults to true. The precision argument is optional and defaults to 1.

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);

图例

图例显示了栅格数据的 nband 属性。exclude_nodata_value 属性，指定是否排除 nodata 值。exclude_nodata_value 属性默认为 true。sample_percent 属性，指定采样百分比，范围从 0 到 1。2.2.0 版本新增。

图例

```
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;
```

20

图例

[ST_Count](#), [ST_SummaryStats](#), [ST_SetBandNoDataValue](#)

11.9.3 ST_Histogram

ST_Histogram — 返回 (bin; 数组) 的直方图。该函数返回一个包含两个元素的数组：第一个元素是 bin 的数组，第二个元素是频率的数组。

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

返回结果包含 min, max, count, percent 信息。返回结果包含 nband 1 信息。



Note

如果 exclude_nodata_value 为 true，则返回结果将排除 nodata 值。如果 exclude_nodata_value 为 false，则返回结果将包含 nodata 值。

width width: 返回结果包含 width 信息。返回结果包含 width 信息，width 信息

返回结果包含 width 信息，width 信息

bins (breakout) 返回结果包含 bins 信息。返回结果包含 bins 信息。

right 返回结果包含 right 信息 (true) 返回结果包含 right 信息。X 返回结果包含 [a, b] 信息 (a, b) 信息。

Changed: 3.1.0 Removed ST_Histogram(table_name, column_name) variant.
2.0.0 返回结果包含 bins 信息。

返回结果包含 bins 信息 - 返回 1, 2, 3 返回结果包含 bins 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

例: 对 2 个波段生成 6 个直方图。

```
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)

例

[ST_Count](#), [ST_SummaryStats](#), [ST_SummaryStatsAgg](#)

11.9.4 ST_Quantile

ST_Quantile — 对 (population) 生成 (quantile) 个分位数。例如, 生成 25%, 50%, 75% 分位数 (percentile) 个分位数。

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);
```



人口 (population) 分位数 (quantile) 百分数 (percentile)
 25%, 50%, 75% 百分数 (percentile)



Note

exclude_nodata_value ☐☐☐☐☐☐☐☐, NODATA ☐☐☐☐☐☐☐☐☐☐.

Changed: 3.1.0 Removed ST_Quantile(table name, column_name) variant.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.



```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Example will consider only pixels of band 1 that are not 249 and in named quantiles --
```

```
SELECT (pvq).*
FROM (SELECT ST_Quantile(rast, ARRAY[0.25,0.75]) As pvq
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvq).quantile;
```

```
quantile | value
-----+-----
0.25    |    253
0.75    |    254
```

```
SELECT ST_Quantile(rast, 0.75) As value
FROM dummy rast WHERE rid=2;
```

254

Real live example. Quantile of all pixels in band 2 intersecting a geometry.

```
SELECT rid, (ST_Quantile(rast,2)).* As pvc
FROM o_4_boston
WHERE ST_Intersects(
    rast,
    ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 892151,224486 892151))',26986)
)
ORDER BY value, quantile,rid
;
```

rid	quantile	value
1	0	0
2	0	0
14	0	1
15	0	2
14	0.25	37
1	0.25	42
15	0.25	47
2	0.25	50
14	0.5	56

1	0.5	64
15	0.5	66
2	0.5	77
14	0.75	81
15	0.75	87
1	0.75	94
2	0.75	106
14	1	199
1	1	244
2	1	255
15	1	255

[ST_Count](#), [ST_SummaryStats](#), [ST_SummaryStatsAgg](#), [ST_SetBandNoDataValue](#)

11.9.5 ST_SummaryStats

`ST_SummaryStats` — Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed if no band is specified.

Synopsis

summarystats **ST_SummaryStats**(raster rast, boolean exclude_nodata_value);
summarystats **ST_SummaryStats**(raster rast, integer nband, boolean exclude_nodata_value);

summarystats returns count, sum, mean, stddev, min, max for the specified raster band. If nband is 1, the first band is used.



Note
If exclude_nodata_value is true, the summary statistics are calculated excluding the nodata value.



Note
If sample_percent is 1, the summary statistics are calculated using the entire raster.

2.2.0 `ST_SummaryStats(rastertable, rastercolumn, ...)` returns summarystats. [ST_SummaryStatsAgg](#) is also available.

2.0.0 `ST_SummaryStats` is available.

Example 1: Summary statistics

```
SELECT rid, band, (stats).*
FROM (SELECT rid, band, ST_SummaryStats(rast, band) As stats
      FROM dummy_rast CROSS JOIN generate_series(1,3) As band
      WHERE rid=2) As foo;
```

rid	band	count	sum	mean	stddev	min	max
2	1	23	5821	253.086957	1.248061	250	254
2	2	25	3682	147.28	59.862188	78	254
2	3	25	3290	131.6	61.647384	62	254

Example 2: Summary statistics by building

PostGIS 64 bit raster data (102,000 x 150x150 x 134,000 x) 574 buildings.

```
WITH
-- our features of interest
feat AS (SELECT gid As building_id, geom_26986 As geom FROM buildings AS b
        WHERE gid IN(100, 103,150)
        ),
-- clip band 2 of raster tiles to boundaries of builds
-- then get stats for these clipped regions
b_stats AS
  (SELECT building_id, (stats).*
   FROM (SELECT building_id, ST_SummaryStats(ST_Clip(rast,2,geom)) As stats
         FROM aerials.boston
         INNER JOIN feat
         ON ST_Intersects(feat.geom,rast)
        ) As foo
  )
-- finally summarize stats
SELECT building_id, SUM(count) As num_pixels,
      MIN(min) As min_pval,
      MAX(max) As max_pval,
      SUM(mean*count)/SUM(count) As avg_pval
FROM b_stats
WHERE count
> 0
GROUP BY building_id
ORDER BY building_id;
```

building_id	num_pixels	min_pval	max_pval	avg_pval
100	1090	1	255	61.0697247706422
103	655	7	182	70.5038167938931
150	895	2	252	185.642458100559

Example 3: Summary statistics

This example returns statistics for each band.

```
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band) As stats
      FROM generate_series(1,3) As band) As foo;
```

band	count	sum	mean	stddev	min	max
1	8450000	725799	82.7064349112426	45.6800222638537	0	255
2	8450000	700487	81.4197705325444	44.2161184161765	0	255
3	8450000	575943	74.682739408284	44.2143885481407	0	255

For a table -- will get better speed if set sampling to less than 100%. Here we set to 25% and get a much faster answer.

```
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band,true,0.25) As stats
      FROM generate_series(1,3) As band) As foo;
```

band	count	sum	mean	stddev	min	max
1	2112500	180686	82.6890480473373	45.6961043857248	0	255
2	2112500	174571	81.448503668639	44.2252623171821	0	255
3	2112500	144364	74.6765884023669	44.2014869384578	0	255

summarystats, ST_SummaryStatsAgg, ST_Count, ST_Clip

11.9.6 ST_SummaryStatsAgg

ST_SummaryStatsAgg — Aggregate. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a set of raster. Band 1 is assumed if no band is specified.

Synopsis

```
summarystats ST_SummaryStatsAgg(raster rast, integer nband, boolean exclude_nodata_value, double precision sample_percent);
summarystats ST_SummaryStatsAgg(raster rast, boolean exclude_nodata_value, double precision sample_percent);
summarystats ST_SummaryStatsAgg(raster rast, integer nband, boolean exclude_nodata_value);
```



count, sum, mean, stddev, min, max **summarystats**. **nbands** 1 **dtype**.



Note!

Note

`exclude nodata value`



Note!

Note

note: `sample_percent` is a float between 0 and 1, representing the percentage of the dataset to sample.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.

```
WITH foo AS (  
  SELECT  
    rast.rast  
  FROM (  
    SELECT ST_SetValue(ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(10, 10, ↵  
      10, 10, 2, 2, 0, 0,0),  
      1, '64BF', 0, 0  
    ),  
    1, 1, 1, -10  
  ),  
  1, 5, 4, 0  
),  
  1, 5, 5, 3.14159  
) AS rast  
) AS rast  
FULL JOIN (  
  SELECT generate_series(1, 10) AS id  
) AS id  
  ON 1 = 1  
)  
SELECT  
  (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)					

[summarystats](#), [ST_SummaryStats](#), [ST_Count](#), [ST_Clip](#)

11.9.7 ST_ValueCount

ST_ValueCount — (integer, integer) Returns the number of pixels in a raster that have a value equal to the search value. The search value can be NULL, in which case the function returns the number of non-NODATA pixels. The search value can also be a string, in which case the function returns the number of pixels that have a value equal to the search value.

Synopsis

SETOF record **ST_ValueCount**(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);

SETOF record **ST_ValueCount**(raster rast, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);

SETOF record **ST_ValueCount**(raster rast, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);

bigint **ST_ValueCount**(raster rast, double precision searchvalue, double precision roundto=0);

bigint **ST_ValueCount**(raster rast, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);

bigint **ST_ValueCount**(raster rast, integer nband, double precision searchvalue, double precision roundto=0);

SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);

SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);

SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);

bigint **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);

bigint **ST_ValueCount**(text rastertable, text rastercolumn, double precision searchvalue, double precision roundto=0);

bigint **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision searchvalue, double precision roundto=0);

例

SELECT ST_ValueCount(rast, 1) AS value, count FROM dummy_rast WHERE rid=2;

SELECT ST_ValueCount(rast, 1) AS value, count FROM dummy_rast WHERE rid=2; searchvalues 250, 251, 252, 253, 254. searchvalues 250, 251, 252, 253, 254.



Note

exclude_nodata_value 为 TRUE 时, NODATA 值将被忽略。

2.0.0 版本中, 该函数返回一个 SETOF 记录。

例

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Example will count only pixels of band 1 that are not 249. --
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast) As pvc
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvc).value;
```

value	count
250	2
251	1
252	2
253	6
254	12

Example will count all pixels of band 1 including 249.

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,1,false) As pvc
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvc).value;
```

value	count
249	2
250	2
251	1
252	2
253	6
254	12

Example will count only non-nodata value pixels of band 2.

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,2) As pvc
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvc).value;
```

value	count
78	1
79	1
88	1
89	1
96	1
97	1
98	1
99	2
112	2

Real live example. Count all the pixels in an aerial raster tile band 2 intersecting a geometry. And return only the pixel band values that have a count > 500.

```
SELECT (pvc).value, SUM((pvc).count) As total
FROM (SELECT ST_ValueCount(rast,2) As pvc
      FROM o_4_boston
      WHERE ST_Intersects(
        rast,
        ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 892151,224486 892151))',26986)
      ) As foo
GROUP BY (pvc).value
HAVING SUM((pvc).count) > 500
ORDER BY (pvc).value;
```

value	total
51	502
54	521

Just return count of pixels in each raster tile that have value of 100 of tiles that intersect a specific geometry.

```
SELECT rid, ST_ValueCount(rast,2,100) As count
FROM o_4_boston
WHERE ST_Intersects(
    rast,
    ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 892151,224486 892151))',26986)
);
```

```
rid | count
----+-----
  1 |    56
  2 |    95
 14 |    37
 15 |    64
```



ST_Count, ST_SetBandNoDataValue

11.10 Raster Inputs

11.10.1 ST_RastFromWKB

ST_RastFromWKB — Return a raster value from a Well-Known Binary (WKB) raster.

Synopsis

```
raster ST_RastFromWKB(bytea wkb);
```



Given a Well-Known Binary (WKB) raster, return a raster.

Availability: 2.5.0

[illegible]

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

ST_RastFromHexWKB — Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.

```
raster ST_RastFromHexWKB(text wkb);
```

Given a Well-Known Binary (WKB) raster in Hex representation, return a raster.
Availability: 2.5.0

[illegible]

upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	
numbands									
0.5	0.5	10	20	2	3	0	0	10	
	0								

ST MetaData, ST RastFromWKB, ST AsBinary/ST AsWKB, ST AsHexWKB

11.11.1 ST AsBinary/ST AsWKB

Synopsis

```
bytea ST_AsBinary(raster rast, boolean outasin=FALSE);
bytea ST_AsWKB(raster rast, boolean outasin=FALSE);
```



```
SELECT ST_AsHexWKB(rast) AS rastbin FROM dummy_rast WHERE rid=1;
```

```
0100000000000000000000000000000000400000000000000008400000000000000 ←
E03F00000000000000E03F000000000000000000000000000000000000000000A0000000A001400
```


ST_RastFromHexWKB, ST_AsBinary/ST_AsWKB

11.11.3 ST_AsGDALRaster

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.

Synopsis

```
bytea ST_AsGDALRaster(raster rast, text format, text[] options=NULL, integer srid=sameasource);
```

[illegible]

- **format** - 指定する形式。GDAL 対応の LibGDAL 形式。例として 'JPEG', 'GTiff', 'PNG' など。GDAL 対応形式は **ST_GDALDrivers** で確認可能。
- **options** - GDAL 設定オプション。例として **GDAL** **EXTRA** など。
- **srs** - proj4text (spatial ref sys テキスト) srtext (spatial ref sys テキスト)。

2.0.0 20200818. GDAL 1.6.0 20100522.

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 创建

```

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;

```

创建

Section [10.3, ST_GDALDrivers, ST_SRID](#)

11.11.4 ST_AsJPEG

ST_AsJPEG — 将栅格数据转换为 JPEG (Joint Photographic Exports Group) 格式 (8 位或 16 位)。该函数返回一个字节数组，包含 1 到 3 个通道。如果输入是 3 通道 RGB 数据，则返回 3 个通道。

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);

```

创建

该函数返回一个字节数组，包含 1 到 3 个通道。如果输入是 3 通道 RGB 数据，则返回 3 个通道。该函数使用 **ST_AsGDALRaster** 函数。该函数返回一个字节数组，包含 1 到 3 个通道。如果输入是 3 通道 RGB 数据，则返回 3 个通道。

- `nband` - 输出带数。默认为 1。
- `nbands` - 输出带数 (JPEG 格式为 3 个带)。指定 RGB 带。指定 `ARRAY[3,2,1]` 指定 3 个带, 指定 2 个带, 指定 1 个带。
- `quality` - 1 到 100 的质量。默认为 90。
- `options` - JPEG 格式 GDAL 驱动程序选项 (`ST_GDALDrivers` 中的 JPEG 格式 `create_options` 选项)。JPEG 格式, 指定 `PROGRESSIVE ON/OFF` 指定 75 到 0 到 100 的质量 `QUALITY` 选项。指定 `GDAL` 格式。

2.0.0 版本。GDAL 1.6.0 版本。

例:

Output the first three bands at 75 percent quality.

```
SELECT ST_AsJPEG(rast) As rastjpg
FROM dummy_rast WHERE rid=2;
```

Output only the first band at 90 percent quality.

```
SELECT ST_AsJPEG(rast,1,90) As rastjpg
FROM dummy_rast WHERE rid=2;
```

Output the first three bands, mapping band 2 to red, band 1 to green, and band 3 to blue, using progressive encoding at 90 percent quality.

```
SELECT ST_AsJPEG(rast,ARRAY[2,1,3],ARRAY['QUALITY=90','PROGRESSIVE=ON']) As rastjpg
FROM dummy_rast WHERE rid=2;
```

例:

Section 10.3, `ST_GDALDrivers`, `ST_AsGDALRaster`, `ST_AsPNG`, `ST_AsTIFF`

11.11.5 ST_AsPNG

`ST_AsPNG` — 将栅格数据转换为 PNG (Portable Network Graphics) 格式 (栅格) 输出。指定 1 到 3 个带, 指定 4 个带。指定 2 到 4 个带。指定 1 个带。指定 RGB 或 RGBA 格式。

Synopsis

```
bytea ST_AsPNG(raster rast, text[] options=NULL);
bytea ST_AsPNG(raster rast, integer nband, integer compression);
bytea ST_AsPNG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsPNG(raster rast, integer[] nbands, integer compression);
bytea ST_AsPNG(raster rast, integer[] nbands, text[] options=NULL);
```

图例

ST_AsPNG(rast) 返回 PNG (Portable Network Graphics) 格式的栅格。ST_AsPNG(rast, options) 返回 PNG 格式的栅格。ST_AsPNG(rast, options, srid) 返回 PNG 格式的栅格，并指定 SRID。ST_AsPNG(rast, options, srid, nband) 返回 PNG 格式的栅格，并指定 SRID 和波段数。

- **nband** - 指定要输出的波段数。
- **nbands** - 指定要输出的波段数 (PNG 格式支持 4 个波段)。指定 RGBA 格式。指定 ARRAY[3,2,1] 指定 3 个波段，指定 2 个波段，指定 1 个波段。
- **compression** - 指定压缩级别 (1 到 9)。指定 1 到 9。
- **options** - PNG 格式选项 GDAL 选项 (ST_GDALDrivers 指定 PNG 格式 create_options 选项)。PNG 格式，指定 ZLEVEL (指定 - 指定 6) 指定，指定 ARRAY['ZLEVEL=9'] 指定。指定 2 指定。指定 GDAL 指定。

2.0.0 指定。GDAL 1.6.0 指定。

图例

```
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;
```

图例

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);
```

图例

ST_AsTIFF(rast) 返回 TIFF (Tagged Image File Format) 格式的栅格。ST_AsTIFF(rast, options) 返回 TIFF 格式的栅格。ST_AsTIFF(rast, options, srid) 返回 TIFF 格式的栅格，并指定 SRID。ST_AsTIFF(rast, options, srid, nband) 返回 TIFF 格式的栅格，并指定 SRID 和波段数。

- **nbands** - 栅格的波段数 (PNG 格式 3 波段). 指定 RGB 波段. 例如 `ARRAY[3,2,1]` 指定 3 波段, 第 2 波段, 第 1 波段. 指定波段顺序.
- **compression** - 格式: JPEG90(栅格), LZW, JPEG, DEFLATE9
- **options** - GTiff 格式 GDAL 选项 (ST_GDALDrivers 格式 GTiff 格式 `create_options` 格式). 指定 GDAL 选项.
- **srid** - 指定 spatial_ref_sys 的 SRID 值. 指定空间参考系统.

2.0.0 版本. GDAL 1.6.0 版本.

格式: JPEG 格式 90%

```
SELECT ST_AsTIFF(rast, 'JPEG90') As rasttiff
FROM dummy_rast WHERE rid=2;
```

格式

ST_GDALDrivers, ST_AsGDALRaster, ST_SRID

11.12 栅格操作

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);
```

格式

指定 geom 的 SRID 值. 指定空间参考系统.

ST_Clip 返回的栅格包含 NODATA 值. NODATA 值指定为 `ST_MinPossibleValue(ST_Band(rast, band))`. 指定 NODATA 值. 指定 NODATA 值. 指定 NODATA 值.

指定 NODATA 选项。 NODATA 选项指定是否将 NODATA 值包含在输出中。 NODATA 选项默认为 true。 NODATA 选项指定是否将 NODATA 值包含在输出中。

If crop is not specified, true is assumed meaning the output raster is cropped to the intersection of the geom and rast extents. If crop is set to false, the new raster gets the same extent as rast. If touched is set to true, then all pixels in the rast 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 版本引入。

版本: 2.1.0 版本引入 C 语言。

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)
CROSS JOIN ST_Clip(r.rast, g.geom, touched =
> true) AS rast_touched
CROSS JOIN 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

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;
```



例: 裁剪影像

Clip all bands of an aerial tile by a 20 meter buffer. Only difference is we don't specify a specific band to clip. So all bands are clipped.

```
SELECT ST_Clip(
    rast,
    ST_Buffer(ST_Centroid(ST_Envelope(rast)), 20),
    false
) from aerials.boston
WHERE rid = 4;
```



例

ST_AddBand, ST_Count, ST_MapAlgebra (callback function version), ST_Intersection

11.12.2 ST_ColorMap

ST_ColorMap — 将 8BUI 栅格 (grayscale, RGB, RGBA) 转换为 4 通道栅格。返回 1 通道栅格。

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);

例

rast 栅格 nband 通道数 colormap 颜色表 8BUI 或 4 通道栅格。colormap 颜色表 8BUI 或 1 通道栅格。

nband 通道数, 或 1 通道。

颜色表, 颜色表 8BUI 或 1 通道 colormap 颜色表。

颜色表 colormap 颜色表:

- grayscale  greyscale - 8BUI  (shades of gray) 
- pseudocolor - 8BUI(RGBA)  4                          
- fire - 8BUI(RGBA)  4                          
- bluered - 8BUI(RGBA)  4                          

colormap (colormap) 5 0 255 (RGBA) 0/100% nodata nv, null nodata

```
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 GDAL (color-relief) gdaldem

☒☒☒ method ☒☒☒:

- INTERPOLATE - 0x00000000000000000000000000000000.
- EXACT - 0x00000000000000000000000000000000. 0x00000000000000000000000000000000 0 0 0 0(RGBA) 0x00000000.
- NEAREST - 0x00000000000000000000000000000000.



Note

ColorBrewer .



Warning

Warning: ST SetBandNoDataValue NODATA NODATA

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.

[illegible]

Setup test raster table.

```

DROP TABLE IF EXISTS funky_shapes;
CREATE TABLE funky_shapes(rast raster);

INSERT INTO funky_shapes(rast)
WITH ref AS (
    SELECT ST_MakeEmptyRaster(200, 200, 0, 200, 1, -1, 0, 0) AS rast
)
SELECT

```

```

    ST_Union(rast)
FROM (
    SELECT
        ST_AsRaster(ST_Rotate(ST_Buffer(ST_GeomFromText('LINESTRING(0 2,50 50,150 150,125 50)'),
            i*2
        ),
        pi() * i * 0.125, ST_Point(50,50)
    ),
    ref.rast, '8BUI'::text, i * 5
    ) AS rast
    FROM ref
    CROSS JOIN generate_series(1, 10, 3) AS i
) AS shapes;

```

```

SELECT
    ST_NumBands(rast) As n_orig,
    ST_NumBands(ST_ColorMap(rast,1, 'greyscale')) As ngrey,
    ST_NumBands(ST_ColorMap(rast,1, 'pseudocolor')) As npseudo,
    ST_NumBands(ST_ColorMap(rast,1, 'fire')) As nfire,
    ST_NumBands(ST_ColorMap(rast,1, 'bluered')) As nbluered,
    ST_NumBands(ST_ColorMap(rast,1, '
100% 255  0  0
80% 160  0  0
50% 130  0  0
30%  30  0  0
20%  60  0  0
0%   0  0  0
nv 255 255 255
')) As nred
FROM funky_shapes;

```

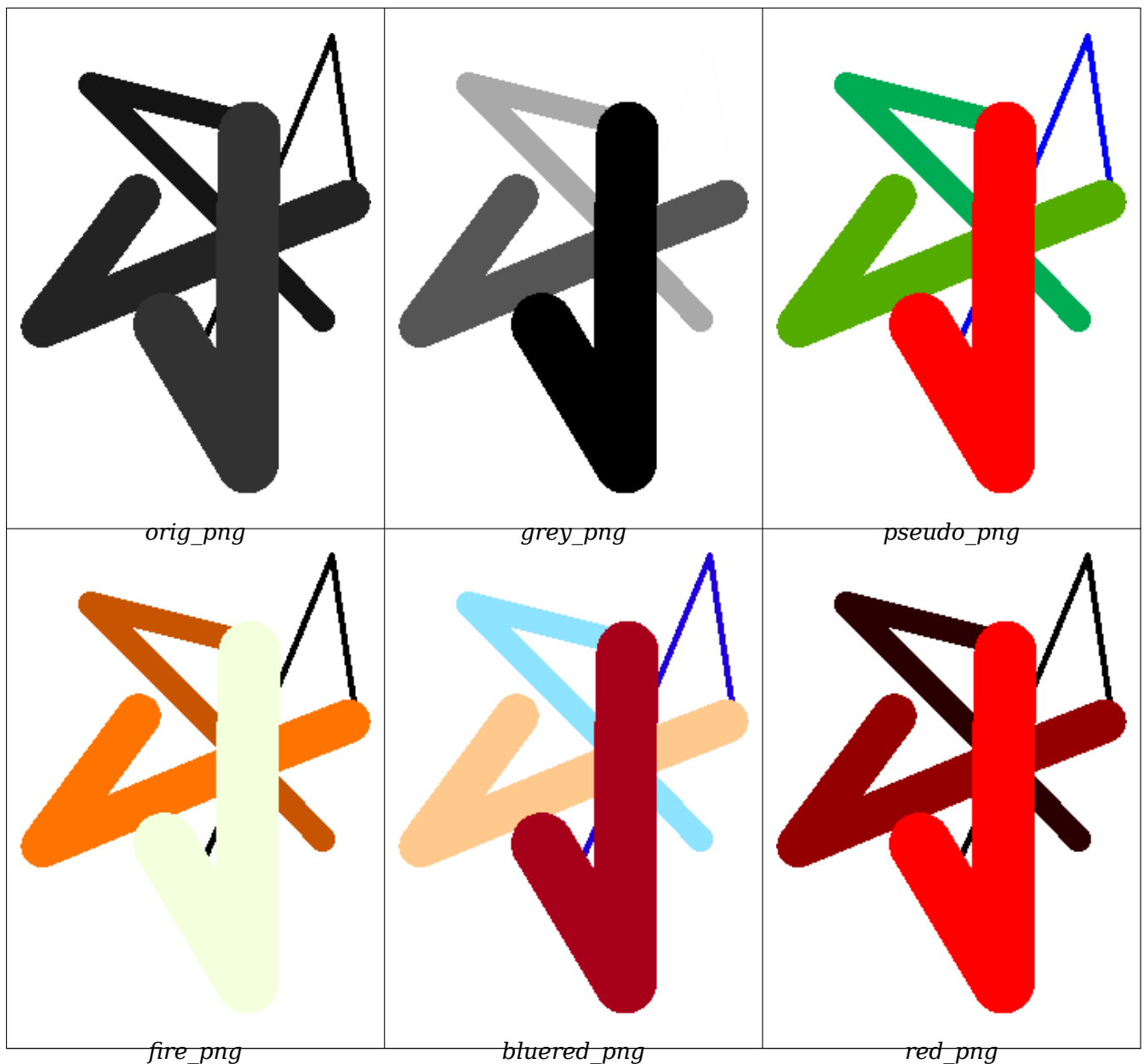
n_orig	ngrey	npseudo	nfire	nbluered	nred
1	1	4	4	4	3

图例: 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);

注意

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: $0.2989 * \text{RED} + 0.5870 * \text{GREEN} + 0.1140 * \text{BLUE}$

Availability: 2.5.0

注意: 注意 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

注意: 注意 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;

```

图 11.12.3

[ST_AsPNG](#), [ST_Reclass](#), [ST_ColorMap](#)

11.12.4 ST_Intersection

ST_Intersection — 返回两个几何体（点、线、面）的交集。如果两个几何体不相交，则返回空集。

Synopsis

```

setof geomval ST_Intersection(geometry geom, raster rast, integer band_num=1);
setof geomval ST_Intersection(raster rast, geometry geom);
setof geomval ST_Intersection(raster rast, integer band, geometry geom);
raster ST_Intersection(raster rast1, raster rast2, double precision[] nodataval);
raster ST_Intersection(raster rast1, raster rast2, text returnband, double precision[] nodataval);
raster ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval);
raster ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval);

```

图 11.12.4

图 11.12.4 展示了 ST_Intersection 函数的使用。图中显示了两个相交的多边形，以及它们交集的结果。

geomval **ST_Intersection**(geometry geom, geometry geom2) 返回两个几何体的交集。如果两个几何体不相交，则返回空集。 (ST_DumpAsPolygon 返回的 geomval 是 geometry 类型，因此 ST_Intersection(geometry, geometry) 返回的也是 geometry 类型。如果返回的几何体是空集，则返回 NULL。WHERE 子句中 ST_Intersection 返回的几何体是空集，则 WHERE 子句中的条件为假。

geomval **ST_Intersection**(raster rast, geometry geom) 返回几何体与栅格的交集。返回的几何体是 geometry 类型。 (ST_Intersection(rast, geom)).geom


```

        (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	

例

[geomval](#), [ST_Intersects](#), [ST_MapAlgebraExpr](#), [ST_Clip](#), [ST_AsText](#)

11.12.5 ST_MapAlgebra (callback function version)

ST_MapAlgebra (callback function version) — 指定した 1 つのラスタ、ラスタセット、ラスタセットの 1 つのバンドを指定した関数で処理する。

Synopsis

```

raster ST_MapAlgebra(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixelftype=NULL,
text extenttype=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0,
text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer[] nband, regprocedure callbackfunc, text pixelftype=NULL,
text extenttype=FIRST, raster customextent=NULL, integer distancex=0, integer distancey=0, text[]
VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, text pixelftype=NULL,
text extenttype=FIRST, raster customextent=NULL, integer distancex=0, integer distancey=0, text[]
VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast1, integer nband1, raster rast2, integer nband2, regprocedure call-
backfunc, text pixelftype=NULL, text extenttype=INTERSECTION, raster customextent=NULL, inte-
ger distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, float8[] mask, boolean
weighted, text pixelftype=NULL, text extenttype=INTERSECTION, raster customextent=NULL, text[]
VARIADIC userargs=NULL);

```

例

指定した 1 つのラスタ、ラスタセット、ラスタセットの 1 つのバンドを指定した関数で処理する。

rast, rast1, rast2, rastbandargset 指定した (代数) 関数で処理するラスタ

rastbandargset ラスタセットのバンド番号のリスト。指定したラスタセットのバンド番号のリスト。指定したラスタセットのバンド番号のリスト。

nband, nband1, nband2 整数。nband 输出栅格的带数。nband1 和 nband2 是输入栅格 rast1 和 rast2 的带数。rast1 和 rast2 是 2 维的栅格。

callbackfunc The callbackfunc parameter must be the name and signature of an SQL or PL/pgSQL function, cast to a regprocedure. An example PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION sample_callbackfunc(value double precision[][][], position integer[][], VARIADIC userargs text[])
RETURNS double precision
AS $$
BEGIN
    RETURN 0;
END;
$$ LANGUAGE 'plpgsql' IMMUTABLE;
```

The callbackfunc must have two or three arguments: a 3-dimension double precision array, a 2-dimension integer array and, optionally, a variadic 1-dimension text array. The first argument value is the set of values (as double precision) from all input rasters. The three dimensions (where indexes are 1-based) are: raster #, row y, column x. The second argument position is the set of pixel positions from the output raster and input rasters. The outer dimension (where indexes are 0-based) is the raster #. The position at outer dimension index 0 is the output raster's pixel position. For each outer dimension, there are two elements in the inner dimension for X and Y. The optional third argument userargs is for passing through any user-specified arguments.

Passing a regprocedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'sample_callbackfunc(double precision[], integer[], text[])::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

mask An n-dimensional array (matrix) of numbers used to filter what cells get passed to map algebra call-back function. 0 means a neighbor cell value should be treated as no-data and 1 means value should be treated as data. If weight is set to true, then the values, are used as multipliers to multiple the pixel value of that value in the neighborhood position.

weighted mask 布尔值 (true/false) 是否使用 mask 值作为乘数。mask 是 mask 值的数组。

pixeltype pixeltype 字符串, 输出栅格的像素类型。pixeltype 可以是 NULL, INTERSECTION, UNION, FIRST, CUSTOM, SECOND, LAST。ST_BandPixelType 返回 pixeltype 值。

extenttype extenttype 字符串 INTERSECTION, UNION, FIRST, SECOND, LAST, CUSTOM。FIRST 表示第一个输入栅格的 extent。

customextent extenttype 字符串 CUSTOM, 自定义 customextent 值。范围 1 到 4。

distancex The distance in pixels from the reference cell in x direction. So width of resulting matrix would be 2*distancex + 1. If not specified only the reference cell is considered (neighborhood of 0).

distancey 距离 Y 方向参考单元的距离。结果矩阵的宽度为 2*distancey + 1。如果没有指定，则只考虑参考单元 (邻域为 0)。

userargs If the callbackfunc declares a third argument, that argument is a variadic text array. All trailing text arguments are passed through to the specified callbackfunc, and are contained in the userargs argument.



Note

(`ST_MapAlgebra`) VARIADIC `rastbandarg` `text[]`, PostgreSQL `Query Language (SQL) Functions` “SQL Functions with Variable Numbers of Arguments”.

`1` `rastbandarg` `text[]`. `1` `rastbandarg`.

`2` `3` `rastbandarg` `1` `rastbandarg`. `2` `3` `rastbandarg`.

`4` `rastbandarg` `1` `rastbandarg` `2` `rastbandarg`. `4` `rastbandarg`.

2.2.0 `mask` `text[]`.

2.1.0 `rastbandarg`.

Example 1

`1` `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
```

`1` `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
```

`1` `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

```

Complete example of tiles of a coverage with neighborhood.

```

WITH foo AS (
    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', ←
        1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0) ←
        AS rast UNION ALL
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0) ←
        AS rast UNION ALL

    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10, ←
        0) AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20, ←
        0) AS rast UNION ALL
    SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30, ←
        0) AS rast UNION ALL

    SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100, ←
        0) AS rast UNION ALL
    SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200, ←
        0) AS rast UNION ALL
    SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300, ←
        0) AS rast
)
SELECT
    t1.rid,
    ST_MapAlgebra(ARRAY[ROW(ST_Union(t2.rast), 1)]::rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])':::regprocedure,
        '32BUI',
        'CUSTOM', t1.rast,
        1, 1
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 4
      AND t2.rid BETWEEN 0 AND 8
      AND ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rid, t1.rast

```

Example like the prior one for tiles of a coverage with neighborhood but using only basic SQL features.

```

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_MetaData(rast)),
  (ST_BandMetaData(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
```

图元: 图元 4

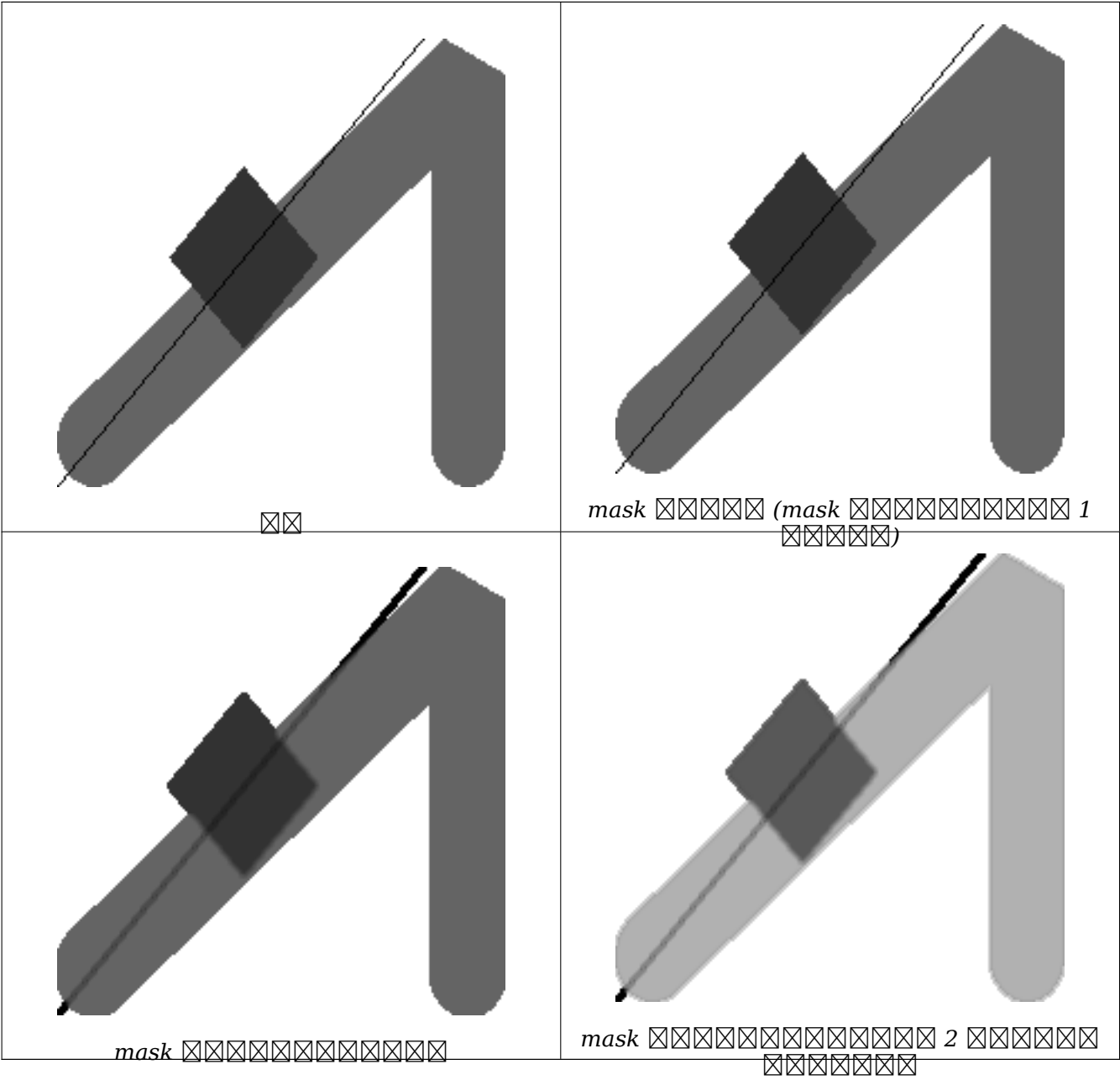
图元 2 图元, 图元 2 图元

```
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, -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
```

图元: mask 图元

```
WITH foo AS (SELECT
  ST_SetBandNoDataValue(ST_SetValue(ST_SetValue(ST_AsRaster(ST_Buffer(ST_GeomFromText('
  LINESTRING(50 50,100 90,100 50)'), 5,'join=bevel'),
  200,200,ARRAY['8BUI'], ARRAY[100], ARRAY[0]), ST_Buffer('POINT(70 70)'::
  geometry,10,'quad_segs=1') ,50),
  'LINESTRING(20 20, 100 100, 150 98)'::geometry,1),0) AS rast )
SELECT 'original' AS title, rast
FROM foo
UNION ALL
SELECT 'no mask mean value' AS title, ST_MapAlgebra(rast,1,'ST_Mean4ma(double precision[],
  int[], text[])'::regprocedure) AS rast
FROM foo
UNION ALL
SELECT 'mask only consider neighbors, exclude center' AS title, ST_MapAlgebra(rast,1,'
  ST_Mean4ma(double precision[], int[], text[])'::regprocedure,
  '{{1,1,1}, {1,0,1}, {1,1,1}}'::double precision[], false) As rast
FROM foo

UNION ALL
SELECT 'mask weighted only consider neighbors, exclude center multi 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;
```



11.12.6

`rastbandarg`, `ST_Union`, `ST_MapAlgebra` (expression version)

11.12.6 ST_MapAlgebra (expression version)

`ST_MapAlgebra` (expression version) — `expression` - `expression` 1 `expression` 2 `expression`, `expression`, `expression`
SQL `expression` 1 `expression` `expression` 1 `expression` `expression`.

Synopsis

raster **ST_MapAlgebra**(raster rast, integer nband, text pixeltype, text expression, double precision nodataval=NULL);

nodata2expr rast2 数据类型 NODATA 数据类型 rast1 数据类型
数据类型, 返回 rast1 数据类型。

nodatanodataval 数据类型 rast1 数据类型 rast2 数据类型 NODATA 数据类型
数据类型。

• expression, nodataexpr 数据类型 nodata2expr 数据类型。

1. [rast1] - rast1 数据类型
2. [rast1.val] - rast1 数据类型
3. [rast1.x] - rast1 数据类型 1-数据类型
4. [rast1.y] - rast1 数据类型 1-数据类型
5. [rast2] - rast2 数据类型
6. [rast2.val] - rast2 数据类型
7. [rast2.x] - rast2 数据类型 1-数据类型
8. [rast2.y] - rast2 数据类型 1-数据类型

图: 图 1 图 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 图 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, 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: 返回栅格表达式返回 PostgreSQL 表达式返回栅格, 返回栅格表达式返回, 返回 1 返回栅格表达式返回栅格。返回栅格表达式返回, 返回 1 返回栅格。

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);

返回



Warning

ST_MapAlgebraExpr 2.1.0 版本已弃用。使用 **ST_MapAlgebra (expression version)** 版本。

`expression` (rast) 返回 `expression` 返回 PostgreSQL 表达式返回栅格, 返回 1 返回栅格表达式返回。 `band` 返回栅格, 返回 1 返回栅格。返回栅格表达式返回, 返回栅格, 返回栅格, 返回 1 返回栅格。

`pixeltype` 返回栅格, 返回栅格表达式返回栅格。 `pixeltype` 返回 NULL 返回, 返回栅格表达式返回 rast 返回栅格表达式返回栅格。

返回栅格表达式返回栅格 [rast], 1-返回栅格表达式返回栅格 [rast.x], 1-返回栅格表达式返回栅格 [rast.y] 返回栅格表达式返回。

2.0.0 版本。

返回

返回 2 返回栅格表达式返回 (modulo) 返回栅格表达式返回 1 返回栅格表达式返回栅格。

```
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

数据库 NODATA 0 数据库 2BUI , 1 数据库

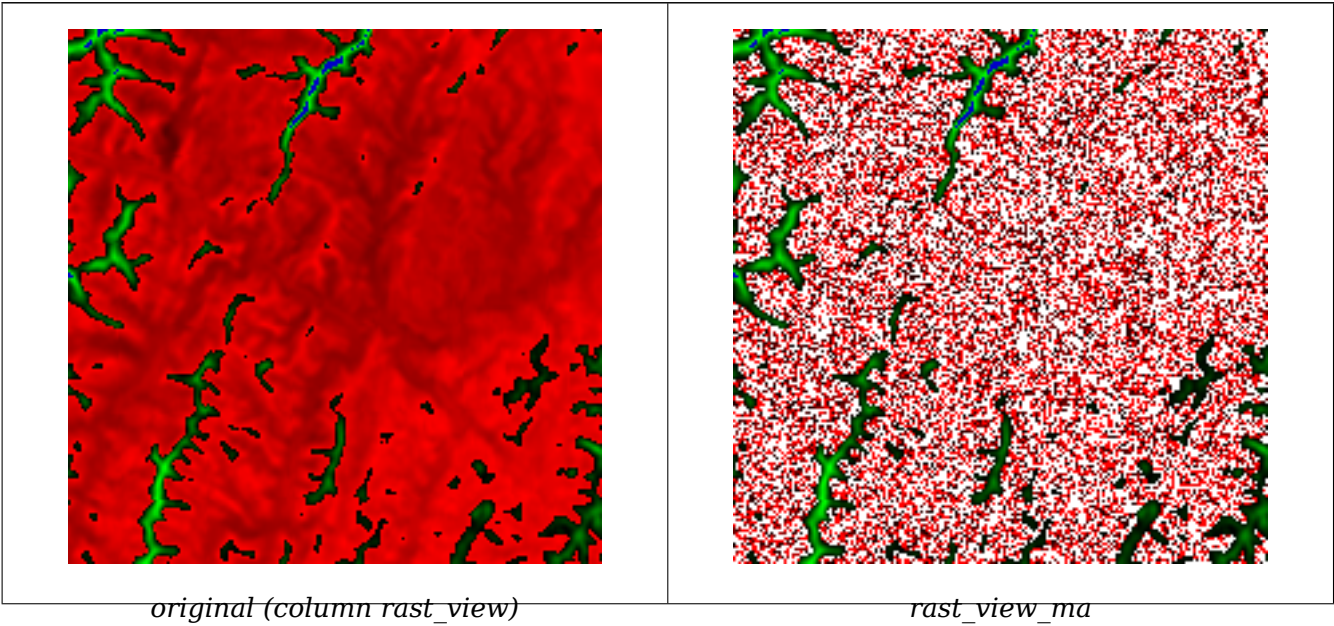
```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
UPDATE dummy_rast SET
  map_rast2 = ST_MapAlgebraExpr(rast,'2BUI'::text,'CASE WHEN [rast] BETWEEN 100 and 250
    THEN 1 WHEN [rast] = 252 THEN 2 WHEN [rast] BETWEEN 253 and 254 THEN 3 ELSE 0 END'::
    text, '0')
WHERE rid = 2;

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;
```

2BUI



pixeltype 数据类型。使用 **ST_BandPixelType** 函数返回像素类型，
数据类型，NULL 数据类型。使用 **ST_BandPixelType** 函数返回 NULL 数据类型，数据类型
数据类型。

extenttype 数据类型

1. INTERSECTION - 数据类型。
2. UNION - 数据类型。
3. FIRST - 数据类型。
4. SECOND - 数据类型。

nodataexpr rast1 数据类型 NODATA 数据类型 rast2 数据类型
数据类型, 数据类型 rast2 数据类型。

nodata2expr rast2 数据类型 NODATA 数据类型 rast1 数据类型
数据类型, 数据类型 rast1 数据类型。

nodatanodataval 数据类型 rast1 数据类型 rast2 数据类型 NODATA 数据类型
数据类型。

pixeltype 数据类型, 数据类型。使用 **ST_BandPixelType** 函数返回 NULL 数据类型
数据类型, 数据类型 rast1 数据类型。

数据类型 [rast1.val], [rast2.val], 数据类型/rast1.val] [rast1.x],
[rast1.y] 数据类型。

2.0.0 数据类型。

数据类型: 数据类型 2 数据类型

数据类型 2 数据类型 (modulo) 数据类型 1 数据类型。

Create a cool set of rasters.

```

DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

-- Insert some cool shapes around Boston in Massachusetts state plane meters --
INSERT INTO fun_shapes(fun_name, rast)
VALUES ('ref', ST_AsRaster(ST_MakeEnvelope(235229, 899970, 237229, 901930,26986),200,200,'8
BUI',0,0));

INSERT INTO fun_shapes(fun_name,rast)
WITH ref(rast) AS (SELECT rast FROM fun_shapes WHERE fun_name = 'ref' )
SELECT 'area' AS fun_name, ST_AsRaster(ST_Buffer(ST_SetSRID(ST_Point(236229, 900930),26986)
, 1000),
ref.rast,'8BUI', 10, 0) As rast
FROM ref
UNION ALL
SELECT 'rand bubbles',
ST_AsRaster((SELECT ST_Collect(geom)
FROM (SELECT ST_Buffer(ST_SetSRID(ST_Point(236229 + i*random()*100, 900930 + j*random()
*100),26986), random()*20) As geom
FROM generate_series(1,10) As i, generate_series(1,10) As j
) As foo ), ref.rast,'8BUI', 200, 0)
FROM ref;

--map them -
SELECT ST_MapAlgebraExpr(area.rast, bub.rast, '[rast2.val]', '8BUI', 'INTERSECTION', '[
rast2.val]', '[rast1.val]') As interrast,
```

```

        ST_MapAlgebraExpr(area.rast, bub.rast, '[rast2.val]', '8BUI', 'UNION', '[rast2.val <
        ]', '[rast1.val]') As unionrast
FROM
    (SELECT rast FROM fun_shapes WHERE
     fun_name = 'area') As area
CROSS JOIN (SELECT rast
FROM fun_shapes WHERE
 fun_name = 'rand bubbles') As bub

```

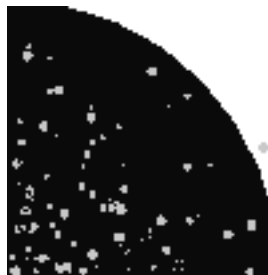


图 10.10 (intersection)

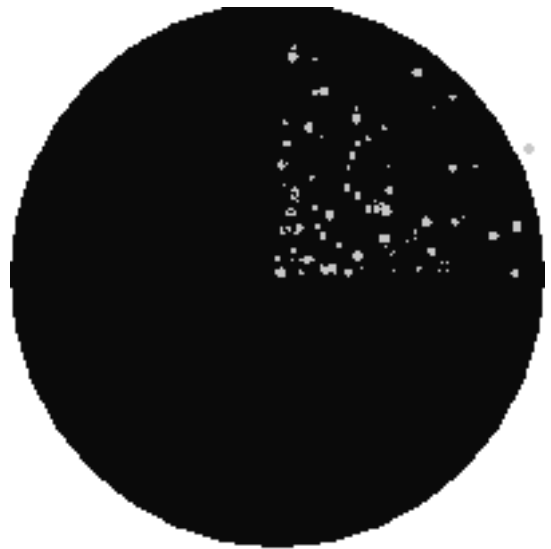


图 10.11 (union)

图 10.12: 图 10.10 和 图 10.11 的并集

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;

```

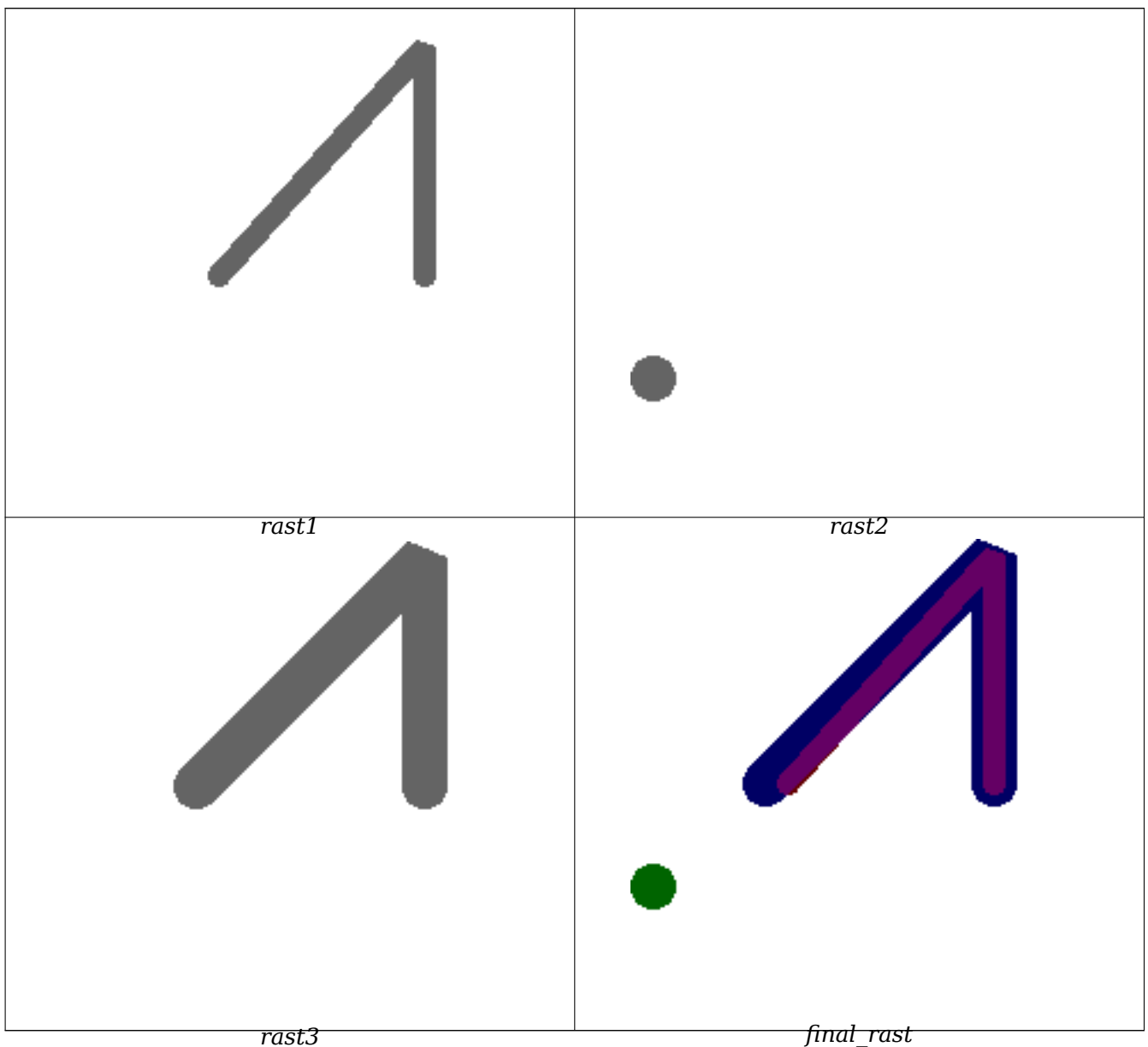


圖 2: 使用 ST_MapAlgebraExpr 函數創建 3 帶光栅

創建新的 3 帶光栅，由前兩個剪切的帶組成，並將第三個帶與我們的幾何圖形疊加。此查詢在 PostGIS 64 位安裝窗口中耗時 3.6 秒。

```

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;

```



￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼.

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 — 1 0: PostgreSQL 9.6.0
 1, 1 0, 1 0. 1 0, 1 0.
 1.

Synopsis

```

raster ST_MapAlgebraFct(raster rast, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster ST_MapAlgebraFct(raster rast, text pixeltype, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster ST_MapAlgebraFct(raster rast, integer band, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, integer band, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster ST_MapAlgebraFct(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);

```



Warning

ST_MapAlgebraFct 2.1.0                                           

raster (rast) raster onerasteruserfunc raster PostgreSQL raster, raster 1
 raster. band raster, raster 1 raster. raster, raster, raster 1 raster.

```
pixeltype 0x00000000, 0x00000000000000000000000000000000. pixeltype 0x NULL 0x00, 0x0000
0x00000000 rast 0x00000000000000000000000000000000.
```

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

250		0
254		0
254		0

將 NODATA 值 (0) 轉換為 2BUI 值, 1 值。

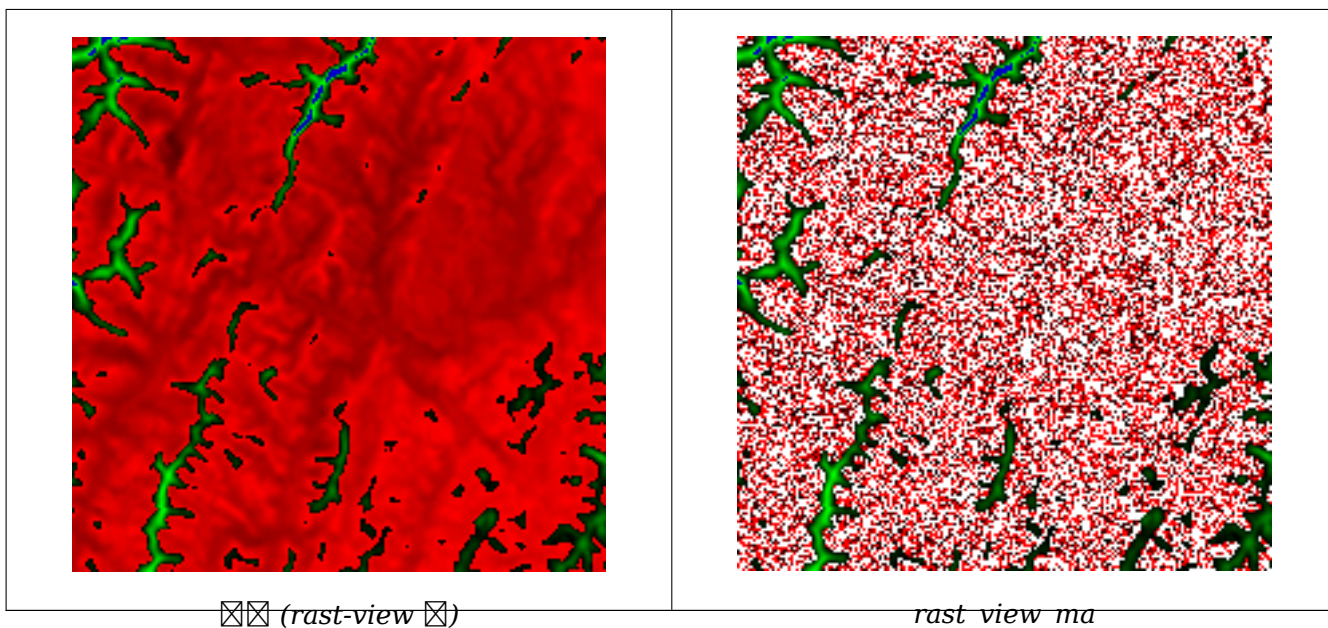
```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
CREATE FUNCTION classify_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
DECLARE
    nodata float := 0;
BEGIN
    IF NOT args[1] IS NULL THEN
        nodata := args[1];
    END IF;
    IF pixel < 251 THEN
        RETURN 1;
    ELSIF pixel = 252 THEN
        RETURN 2;
    ELSIF pixel
> 252 THEN
        RETURN 3;
    ELSE
        RETURN nodata;
    END IF;
END;
$$
LANGUAGE 'plpgsql';
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ↵
[],text[])::regprocedure, '0') WHERE rid = 2;

SELECT DISTINCT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 5) AS i CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;

origval | mapval
-----+-----
    249 |      1
    250 |      1
    251 |
    252 |      2
    253 |      3
    254 |      3

SELECT ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast WHERE rid = 2;

b1pixtyp
-----
2BUI
```



3 个参数: 第一个参数是像素值, 第二个参数是位置, 第三个参数是变长参数列表, 用于指定要应用的函数。 3 个参数: 第一个参数是像素值, 第二个参数是位置。

```
CREATE FUNCTION rast_plus_tan(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
BEGIN
    RETURN tan(pixel) * pixel;
END;
$$
LANGUAGE 'plpgsql';

SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(rast_view),
    ST_MapAlgebraFct(rast_view,1,NULL,'rast_plus_tan(float,integer[],text[])'::
        regprocedure)
    ),
    ST_Band(rast_view,2)
),
    ST_Band(rast_view, 3) As rast_view_ma
)
FROM wind
WHERE rid=167;
```

图 11.12.10

[ST_MapAlgebraExpr](#), [ST_BandPixelType](#), [ST_GeoReference](#), [ST_SetValue](#)

11.12.10 ST_MapAlgebraFct

ST_MapAlgebraFct — 对 2 个栅格: 对 2 个栅格应用 PostgreSQL 函数。 对 1 个栅格应用函数, 对 1 个栅格应用函数。 对 1 个栅格应用函数, 对 1 个栅格应用函数。 对 1 个栅格应用函数, 对 1 个栅格应用函数。 INTERSECTION 操作。

Synopsis

raster **ST_MapAlgebraFct**(raster rast1, raster rast2, regprocedure tworastuserfunc, text pixeltypename=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);
 raster **ST_MapAlgebraFct**(raster rast1, integer band1, raster rast2, integer band2, regprocedure tworastuserfunc, text pixeltypename=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);

注意



Warning

ST_MapAlgebraFct 2.1.0 已弃用。请改用 **ST_MapAlgebra (callback function version)**。

`rast1`, `rast2` 是 PostgreSQL 中的栅格，`band1` 和 `band2` 是栅格的波段编号，`band1` 和 `band2` 都是 1 到 255 之间的整数。如果 `band1` 和 `band2` 都是 1，则使用默认值。

`pixeltypename` 是栅格的像素数据类型，可以是 `NULL`。如果 `pixeltypename` 是 `NULL`，则使用 `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 SQL 函数，参见 [SQL Functions with Variable Numbers of Arguments](#)。

**Note**

在调用 `tworastuserfunc` 时，`text[]` 参数是必需的。

2.0.0 版本已弃用。

例： 定义用户自定义函数。

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;

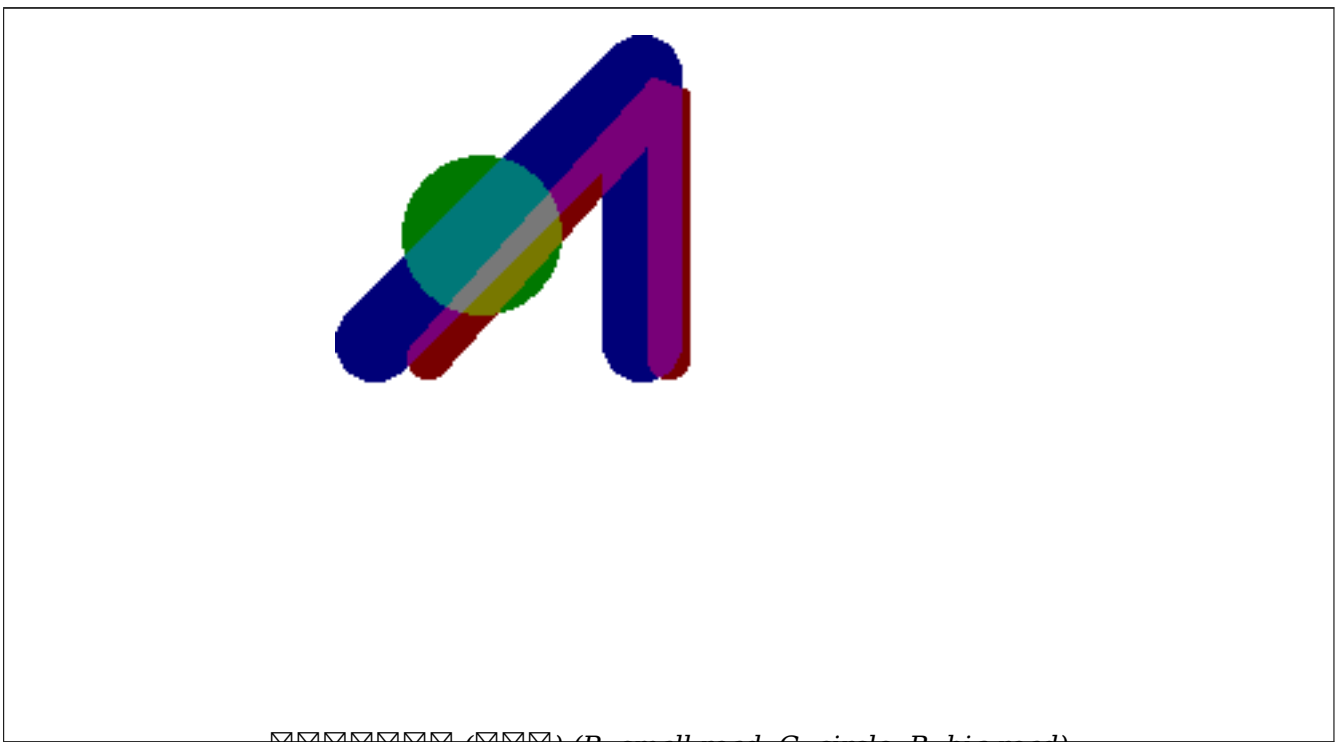
-- prep our test table of rasters
DROP TABLE IF EXISTS map_shapes;
CREATE TABLE map_shapes(rid serial PRIMARY KEY, rast raster, bnum integer, descrip text);
INSERT INTO map_shapes(rast,bnum, descrip)
WITH mygeoms
    AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(90,90),30) As geom, 'circle' As descrip
        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;

```



国国国国国国国国国国国国国国国国国国

```

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)';

```



ST_MapAlgebraFctNgb

ST

[ST_MapAlgebraExpr](#), [ST_BandPixelType](#), [ST_GeoReference](#), [ST_SetValue](#)

11.12.11 ST_MapAlgebraFctNgb

ST_MapAlgebraFctNgb — 返回 1 个值: 返回 PostgreSQL 的最近邻域 (Map Algebra Nearest Neighbor) 函数。该函数使用 PostgreSQL 的最近邻域 (neighborhood) 函数。

Synopsis

raster **ST_MapAlgebraFctNgb**(raster rast, integer band, text pixeltype, integer ngbwidth, integer ngbheight, regprocedure onerastngbuserfunc, text nodatamode, text[] VARIADIC args);

ST



Warning

ST_MapAlgebraFctNgb 2.1.0 已弃用。请改用 **ST_MapAlgebra (callback function version)**。

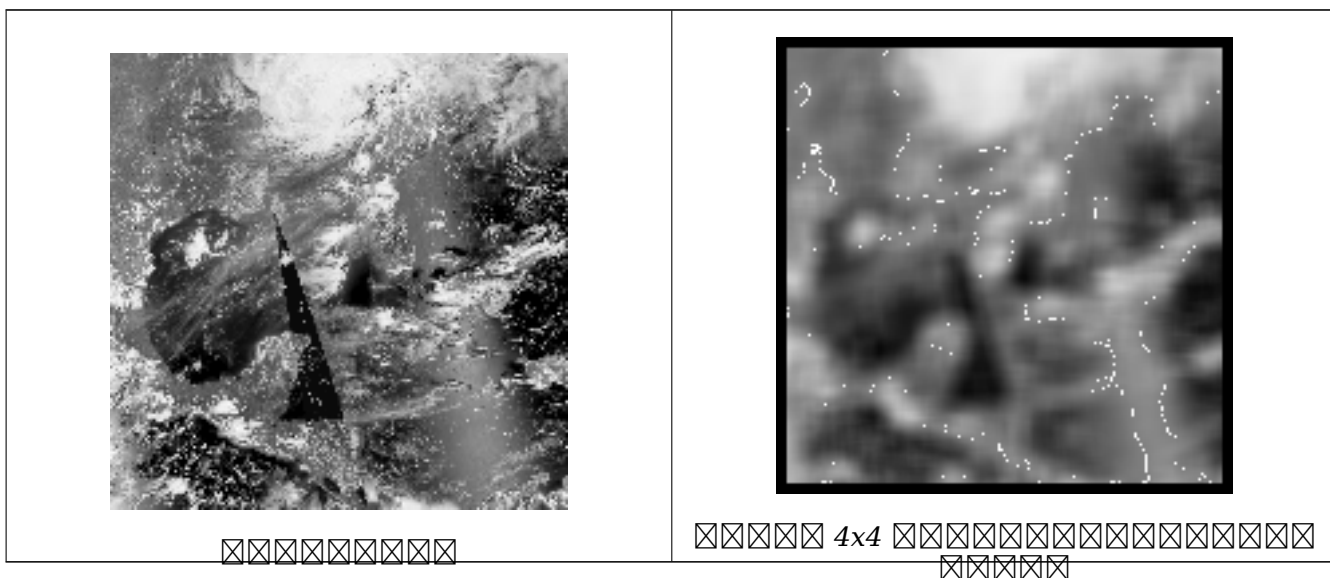
返回 1 个值: 返回 PostgreSQL 的最近邻域 (neighborhood) 函数。该函数使用 PostgreSQL 的最近邻域 (neighborhood) 函数。该函数使用 PostgreSQL 的最近邻域 (neighborhood) 函数。

rast 返回 PostgreSQL 的最近邻域 (neighborhood) 函数。


```

1,
'8BUI',
4,
4,
'rast_avg(float[][], text, text[])::regprocedure,
'NULL',
NULL) As nn_with_border
FROM katrinas_rescaled
limit 1;

```



☐☐

[ST_MapAlgebraFct](#), [ST_MapAlgebraExpr](#), [ST_Rescale](#)

11.12.12 ST_Reclass

ST_Reclass — 对输入栅格进行重新分类。 `nband` 指定要处理的波段数量。 `nband` 为 1 时，默认处理所有波段。 `reclassarg` 指定重新分类的表达式。 `reclassarg` 可以是 16BUI 或 8BUI 格式。

Synopsis

```

raster ST_Reclass(raster rast, integer nband, text reclassexpr, text pixeltype, double precision no-
dataval=NULL);
raster ST_Reclass(raster rast, reclassarg[] VARIADIC reclassargset);
raster ST_Reclass(raster rast, text reclassexpr, text pixeltype);

```

☐☐

Creates a new raster formed by applying a reclassification operation defined by the `reclassexpr` on the input raster (`rast`). Refer to [reclassarg](#) for the description of reclassification expressions. If no band is specified band 1 is assumed.

The new raster will have the same georeference, width, and height as the original raster. The bands of the new raster have pixel type of pixeltype. If reclassargset is specified then each reclassarg defines the type of the target band. Bands not designated are returned unchanged.

2.0.0

Example: Basic

2 8BUI 4BUI 101 254 NODATA

```
ALTER TABLE dummy_rast ADD COLUMN reclass_rast raster;
UPDATE dummy_rast SET reclass_rast = ST_Reclass(rast,2,'0-87:1-10, 88-100:11-15, 101-254:0-0', '4BUI',0) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast,2,i,j) As origval,
       ST_Value(reclass_rast, 2, i, j) As reclassval,
       ST_Value(reclass_rast, 2, i, j, false) As reclassval_include_nodata
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

col	row	origval	reclassval	reclassval_include_nodata
1	1	78	9	9
2	1	98	14	14
3	1	122		0
1	2	96	14	14
2	2	118		0
3	2	180		0
1	3	99	15	15
2	3	112		0
3	3	169		0

1, 2, 3 1BB, 4BUI, 4BUI. (1BB, 4BUI, 4BUI) reclassarg

```
UPDATE dummy_rast SET reclass_rast =
  ST_Reclass(
    rast,
    ROW(2,'0-87]:1-10, (87-100]:11-15, (101-254]:0-0', '4BUI',NULL)::reclassarg,
    ROW(1,'0-253]:1, 254:0', '1BB', NULL)::reclassarg,
    ROW(3,'0-70]:1, (70-86:2, [86-150):3, [150-255:4', '4BUI', NULL)::reclassarg
  ) WHERE rid = 2;

SELECT i as col, j as row,ST_Value(rast,1,i,j) As ov1, ST_Value(reclass_rast, 1, i, j) As rv1,
       ST_Value(rast,2,i,j) As ov2, ST_Value(reclass_rast, 2, i, j) As rv2,
       ST_Value(rast,3,i,j) As ov3, ST_Value(reclass_rast, 3, i, j) As rv3
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

col	row	ov1	rv1	ov2	rv2	ov3	rv3
1	1	253	1	78	9	70	1
2	1	254	0	98	14	86	3

3	1	253	1	122	0	100	3
1	2	253	1	96	14	80	2
2	2	254	0	118	0	108	3
3	2	254	0	180	0	162	4
1	3	250	1	99	15	90	3
2	3	254	0	112	0	108	3
3	3	254	0	169	0	175	4

32BF

32BF ((8BUI,8BUI,8BUI)

```
ALTER TABLE wind ADD COLUMN rast_view raster;
UPDATE wind
  set rast_view = ST_AddBand(
    NULL,
    ARRAY[
      ST_Reclass(rast, 1, '0.1-10]:1-10,9-10]:11,(11-33:0'::text, '8BUI'::text,0),
      ST_Reclass(rast,1, '11-33):0-255,[0-32:0,(34-1000:0'::text, '8BUI'::text,0),
      ST_Reclass(rast,1,'0-32]:0,(32-100:100-255'::text, '8BUI'::text,0)
    ]
  );
```

ST_AddBand, ST_Band, ST_BandPixelType, ST_MakeEmptyRaster, reclassarg, ST_Value

11.12.13 ST_ReclassExact

ST_ReclassExact — Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.

Synopsis

raster **ST_ReclassExact**(raster rast, double precision[] inputvalues, double precision[] outputvalues, integer bandnumber=1, text pixeltype=32BF, double precision nodatavalue=NULL);

Creates a new raster formed by applying a reclassification operation defined by the inputvalues and outputvalues arrays. Pixel values found in the input array are mapped to the corresponding value in the output array. All other pixel values are mapped to the nodatavalue.

The output pixel type defaults to float, but can be specified using the pixeltype parameter. If no bandnumber is specified band 1 is assumed.

The new raster will have the same georeference, width, and height as the original raster. Bands not designated are returned unchanged.

Availability: 3.6.0

Example: Basic

Create a small raster and map its pixels to new values.

```
CREATE TABLE reclasseexact (
    id integer,
    rast raster
);

--
-- Create a raster with just four pixels
-- [1 2]
-- [3 4]
--
INSERT INTO reclasseexact (id, rast)
SELECT 1, ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(2,    -- width in pixels
    2,    -- height in pixels
    0,    -- upper-left x-coordinate
    0,    -- upper-left y-coordinate
    1,    -- pixel size in x-direction
    -1,   -- pixel size in y-direction (negative for north-up)
    0,    -- skew in x-direction
    0,    -- skew in y-direction
    4326  -- SRID (e.g., WGS 84)
),
    '32BUI'::text, -- pixel type (e.g., '32BF' for float, '8BUI' for unsigned 8-bit int)
    0.0,          -- initial value for the band (e.g., 0.0 or a no-data value)
    -99           -- nodatavalue
),
    1, -- band number (usually 1 for single-band rasters)
    1, -- x origin for setting values (usually 1)
    1, -- y origin for setting values (usually 1)
    ARRAY[
        ARRAY[1, 2],
        ARRAY[3, 4]
    ]::double precision[][] -- 2D array of values
);

-- Reclass the values to new values
-- and dump the values of the new raster for display
WITH rc AS (
    SELECT ST_ReclassExact(rast,          -- input raster
        ARRAY[4,3,2,1],    -- input map
        ARRAY[14,13,12,11], -- output map
        1,                  -- band number to remap
        '32BUI'             -- output raster pixtype
    ) AS rast
    FROM reclasseexact
    WHERE id = 1
)
SELECT 'rce-1', (ST_DumpValues(rc.rast)).*
FROM rc;
```

☒☒

[ST_Reclass](#), [ST_AddBand](#), [ST_Band](#), [ST_MakeEmptyRaster](#)

11.12.14 ST_Union

ST_Union — 将 1 个栅格数据集合并为一个栅格数据集。

Synopsis

```
raster ST_Union(setof raster rast);
raster ST_Union(setof raster rast, unionarg[] unionargset);
raster ST_Union(setof raster rast, integer nband);
raster ST_Union(setof raster rast, text uniontype);
raster ST_Union(setof raster rast, integer nband, text uniontype);
```

注意

将 1 个栅格数据集合并为一个栅格数据集。 unionargset 是 uniontype 的集合。 uniontype 可以是 LAST(默认), FIRST, MIN, MAX, COUNT, SUM, MEAN, RANGE 等。



Note

In order for rasters to be unioned, they must all have the same alignment. Use [ST_SameAlignment](#) and [ST_NotSameAlignmentReason](#) for more details and help. One way to fix alignment issues is to use [ST_Resample](#) and use the same reference raster for alignment.

2.0.0 版本引入。

更新: 2.1.0 版本引入 (C 语言接口)。

2.1.0 版本引入 ST_Union(rast, unionarg) 函数。

更新: 2.1.0 版本引入 ST_Union(rast) 函数 1 个栅格数据集合并为一个栅格数据集。 PostGIS 3.7.0 版本引入。

更新: 2.1.0 版本引入 ST_Union(rast, uniontype) 函数 4 个栅格数据集合并为一个栅格数据集。

示例: 将栅格数据集合并为一个栅格数据集

This creates a single band from first band of raster tiles. That form the original file system tile.

```
SELECT filename, ST_Union(rast,1) As file_rast
FROM sometable WHERE filename IN('dem01','dem02') GROUP BY filename;
```

示例: 将栅格数据集合并为一个栅格数据集

This creates a multi band raster collecting all the tiles that intersect a line. Note that 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 aeriels.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

注意: 此函数返回一个多波段栅格。

此函数返回一个多波段栅格，该栅格包含所有与给定线相交的瓦片。

This creates a multi band raster collecting all the tiles that intersect a line.

```
SELECT ST_Union(rast,ARRAY[ROW(2, 'LAST'), ROW(1, 'LAST'), ROW(3, 'LAST')]::unionarg[])
FROM aeriAls.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

参数

unionarg, **ST_Envelope**, **ST_ConvexHull**, **ST_Clip**, **ST_Union**

11.13 空间函数

11.13.1 ST_Distinct4ma

ST_Distinct4ma — 返回指定栅格中所有非空像素的平均值。

Synopsis

float8 **ST_Distinct4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Distinct4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);

参数

返回指定栅格中所有非空像素的平均值。



Note

从 1.0 开始，**ST_MapAlgebraFctNgb** 函数已弃用。



Note

从 2.0 开始，**ST_MapAlgebra (callback function version)** 函数已弃用。



Warning

2.1.0 版本中，**ST_MapAlgebraFctNgb** 函数已弃用。

2.0.0 版本中，**ST_Distinct4ma** 函数已弃用。

从 2.1.0 版本开始，**ST_Distinct4ma** 函数已弃用。

例

```
WITH foo AS (
  SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ←
    ', NULL::double precision, -9999),
    1, 1, 1, ARRAY[
      [10, 10, 10],
      [10, NULL, 20],
      [20, 20, 20]
    ]::double precision[][])
  ) AS rast
)
SELECT ST_Value(
  ST_MapAlgebra(
    rast,
    1,
    'ST_InvDistWeight4ma(double precision[][][],integer[],text[])::regprocedure,
    '32BF',
    'FIRST',
    NULL,
    1,
    1
  ),
  2, 2
) AS interpolated_value
FROM foo;
```

15

例

ST_MapAlgebra (callback function version), ST_MinDist4ma

11.13.3 ST_Max4ma

ST_Max4ma — 返回栅格中每个像素的局部最大值。

Synopsis

float8 **ST_Max4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Max4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

例

返回栅格中每个像素的局部最大值。

例 2 例, 例 userargs 例 NODATA 例。



Note

例 1 例 **ST_MapAlgebraFctNgb** 例。



Note

从 2.0 开始，`ST_MapAlgebra (callback function version)` 已弃用。请改用 `ST_MapAlgebraFctNgb`。



Warning

2.1.0 版本中，`ST_MapAlgebraFctNgb` 的第二个参数 `1` 已弃用。

2.0.0 版本中，`ST_MapAlgebraFctNgb` 已弃用。

从 2.1.0 版本开始，`ST_MapAlgebraFctNgb` 已弃用。

SQL

```
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)	

SQL

`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 移动平均值的函数。

Synopsis

`float8 ST_Mean4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);`
`double precision ST_Mean4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);`

SQL

计算 4 移动平均值的函数。

从 2.0 开始，`ST_Mean4ma` 的 `userargs` 参数已弃用，请改用 `NODATA` 模式。



Note

1 ST_MapAlgebraFctNgb



Note

2 **ST_MapAlgebra (callback function version)**



Warning

2.1.0 **ST_MapAlgebraFctNgb** 1 .

2.0.0 □□□□□□□□□□.

☒☒☒☒: 2.1.0 ☒☒☒☒☒☒ 2 ☒☒☒☒☒☒☒.

☒☒: ☒☒ 1

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, '32BF', 1, 1, 'ST_Mean4ma(float[][]),text,text[]) ←
        '::regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

```

rid |      st_value
-----+-----
  2 | 253.22229003906
(1 row)

```

☒☒: ☒☒ 2

```
SELECT
  rid,
  ST_Value(
    ST_MapAlgebra(rast, 1, 'ST_Mean4ma(double precision[][][], integer[][], text[])':: ↵
      regprocedure,'32BF', 'FIRST', NULL, 1, 1),
    2,
    2)
FROM dummy_rast
WHERE rid = 2;
```

```

rid |      st_value
-----+-----
    2 | 253.222229003906
(1 row)

```

¶¶

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra](#) (callback function version), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Range4ma](#), [ST_StdDev4ma](#)

11.13.5 ST_Min4ma

[ST_Min4ma](#) — 返回栅格中的最小值。

Synopsis

float8 **ST_Min4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Min4ma**(double precision[][] value, integer[][] pos, text[] VARIADIC userargs);

¶¶

返回栅格中的最小值。

¶¶ 2 个参数, 用户参数 userargs 可以指定 NODATA 值。



Note

¶¶ 1 个 [ST_MapAlgebraFctNgb](#) 函数。



Note

¶¶ 2 个 [ST_MapAlgebra](#) (callback function version) 函数。



Warning

2.1.0 版本 [ST_MapAlgebraFctNgb](#) 函数 1 个参数。

2.0.0 版本。

更新: 2.1.0 版本 2 个参数。

¶¶

```
SELECT
  rid,
  ST_Value(ST_MapAlgebraFctNgb(rast, 1, NULL, 1, 1, 'ST_Min4ma(float[][],text,text[])'::
    regprocedure, 'ignore', NULL), 2, 2
)
FROM dummy_rast
WHERE rid = 2;
```

```
rid | st_value
-----+-----
  2 |      250
(1 row)
```

☐☐

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra](#) (callback function version), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Mean4ma](#), [ST_Range4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

11.13.6 ST_MinDist4ma

`ST_MinDist4ma` — 返回具有指定位置 (点、线、面) 的栅格的最小距离值。

Synopsis

double precision **ST_MinDist4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

☐☐

返回具有指定位置 (点、线、面) 的栅格的最小距离值。



Note

返回具有指定位置 [ST_InvDistWeight4ma](#) 的栅格的最小距离值。返回具有指定位置 [ST_InvDistWeight4ma](#) 的栅格的最小距离值。



Note

返回具有指定位置 [ST_MapAlgebra](#) (callback function version) 的栅格的最小距离值。

2.1.0 版本引入。

☐☐

```
WITH foo AS (  
  SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ←  
    ', NULL::double precision, -9999),  
    1, 1, 1, ARRAY[  
      [10, 10, 10],  
      [10, NULL, 20],  
      [20, 20, 20]  
    ]::double precision[][])  
  ) AS rast  
)  
SELECT ST_Value(  
  ST_MapAlgebra(  
    rast,  
    1,  
    'ST_MinDist4ma(double precision[][][],integer[][],text[])':::regprocedure,  
    '32BF',  
    'FIRST',  
    NULL,
```

```
1,
1
),
2, 2
) AS min_distance
FROM foo;
```

1

图例

[ST_MapAlgebra \(callback function version\)](#), [ST_InvDistWeight4ma](#)

11.13.7 ST_Range4ma

ST_Range4ma — 返回 4 邻域范围内的聚合值。

Synopsis

float8 **ST_Range4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Range4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

图例

返回 4 邻域范围内的聚合值。

参数 2 个，userargs 参数 NODATA 返回 NULL。



Note

参数 1 是 [ST_MapAlgebraFctNgb](#) 返回的聚合值。



Note

参数 2 是 [ST_MapAlgebra \(callback function version\)](#) 返回的聚合值。



Warning

2.1.0 版本中 [ST_MapAlgebraFctNgb](#) 返回的聚合值 1 是聚合值。

2.0.0 版本中返回 NULL。

图例: 2.1.0 版本中返回 2 个聚合值。

例

```
SELECT
  rid,
  ST_Value(ST_MapAlgebraFctNgb(rast, 1, NULL, 1, 1, 'ST_Range4ma(float[][],text,text[]) ←
    '::regprocedure, 'ignore', NULL), 2, 2
)
FROM dummy_rast
WHERE rid = 2;
```

rid	st_value
2	4
(1 row)	

例

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra \(callback function version\)](#), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Mean4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

11.13.8 ST_StdDev4ma

[ST_StdDev4ma](#) — 计算 4 邻域像素的标准差。

Synopsis

float8 **ST_StdDev4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_StdDev4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);

例

计算 4 邻域像素的标准差。



Note

例 1 [ST_MapAlgebraFctNgb](#) 计算 4 邻域像素的标准差。



Note

例 2 [ST_MapAlgebra \(callback function version\)](#) 计算 4 邻域像素的标准差。



Warning

2.1.0 之前 [ST_MapAlgebraFctNgb](#) 计算 4 邻域像素的标准差。

2.0.0 之前 计算 4 邻域像素的标准差。

例: 2.1.0 之前 计算 2 邻域像素的标准差。

例

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, '32BF', 1, 1, 'ST_StdDev4ma(float[][],text,text ↵
        [])'::regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

rid	st_value
2	1.30170822143555

(1 row)

例

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra](#) (callback function version), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Mean4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

11.13.9 ST_Sum4ma

[ST_Sum4ma](#) — 返回指定栅格中指定位置的 4 邻域像素之和。

Synopsis

float8 **ST_Sum4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Sum4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);

例

返回指定栅格中指定位置的 4 邻域像素之和。
例 2 例, 例 userargs 例 NODATA 例。



Note

例 1 [ST_MapAlgebraFctNgb](#) 例。



Note

例 2 [ST_MapAlgebra](#) (callback function version) 例。



Warning

2.1.0 例 [ST_MapAlgebraFctNgb](#) 例 1 例。

2.0.0 例。
例: 2.1.0 例 2 例。

例

```
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)
```

例

[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` — 返回指定栅格 (栅格) 的方位角。方位角以度为单位。

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);

例

`ST_Aspect` (栅格) 返回方位角。方位角以度为单位。
 units 可以是 RADIANS, DEGREES(度)。
 units = RADIANS 时，方位角范围是 0 到 2π 。
 units = DEGREES 时，方位角范围是 0 到 360。
 方位角范围是 0 到 360，负值表示 -1 度。



Note

方位角 (slope), 方位角 (aspect), 方位角 (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，方位角函数返回的是方位角。

更新: 2.1.0 版本中，`ST_MapAlgebra` 函数，`interpolate_nodata` 参数默认为 `FALSE`。

更新: 2.1.0 版本中，方位角函数返回的是方位角。2.1.0 版本中，方位角函数返回的是方位角。

图例: 图 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)

```

图例: 图 2

Complete example of tiles of a coverage.

```

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;

```

更新

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_HillShade, ST_Slope

11.14.2 ST_HillShade

ST_HillShade — 地形陰影を生成する関数。詳細は [ST_HillShade](#) を参照。

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);

更新

azimuth 方位角。0 から 360 度の範囲で指定する。0 は北を指す。単位は度。

altitude 傾斜角。0 から 90 度の範囲で指定する。0 は水平面を指す。単位は度。

max_bright 最大明るさ。0 から 255 の範囲で指定する。0 は黒を指す。単位は 255。

scale スケール。1 から 370400 の範囲で指定する。1 は 1:1 のスケールを指す。単位は 370400。

interpolate_nodata 無データ値を補間するかどうか。true: 補間する。false: 補間しない。ST_InvDistWeight4ma 関数と同様に、NODATA 値は補間されない。



Note

地形陰影の仕組みについては、[How hillshade works](#) を参照してください。

2.0.0 更新履歴

更新: 2.1.0 更新 ST_MapAlgebra() 関数。interpolate_nodata 引数を追加。

更新: 2.1.0 更新 ST_HillShade() 関数。2.1.0 更新 ST_HillShade() 関数。

更新: 1

```
WITH foo AS (
  SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF',
    1, 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1],
      [1, 2, 2, 2, 1],
      [1, 2, 3, 2, 1],
    ]
  )
```

```

        [1, 2, 2, 2, 1],
        [1, 1, 1, 1, 1]
    ]::double precision[][])
    ) AS rast
)
SELECT
    ST_DumpValues(ST_HillShade(rast, 1, '32BF'))
FROM foo

```

```

-----
(1,"{NULL,NULL,NULL,NULL,NULL},{NULL,251.32763671875,220.749786376953,147.224319458008, ←
NULL},{NULL,220.749786376953,180.312225341797,67.7497863769531,NULL},{NULL ←
,147.224319458008,
67.7497863769531,43.1210060119629,NULL},{NULL,NULL,NULL,NULL,NULL}}")
(1 row)

```

图例: 图 2

Complete example of tiles of a coverage.

```

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;

```

图例

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” 粗糙度 (roughness)” 粗糙度。

Synopsis

raster **ST_Roughness**(raster rast, integer nband, raster customextent, text pixeltype="32BF", boolean interpolate_nodata=FALSE);

描述

计算给定 DEM 的粗糙度。

2.1.0 版本引入。

SQL

```
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, 2, 3, 3],
      [1, 2, 4, 15, 3],
      [2, 4, 10, 7, 5],
      [3, 5, 7, 6, 4],
      [3, 3, 5, 4, 4]
    ]::double precision[][])
  ) AS rast
)
SELECT ST_Value(ST_Roughness(rast, 1, '32BF'), 3, 3) AS roughness
FROM foo;
```

13

相关链接

[ST_MapAlgebra \(callback function version\)](#), [ST_TRI](#), [ST_TPI](#), [ST_Slope](#), [ST_HillShade](#), [ST_Aspect](#)

11.14.4 ST_Slope

ST_Slope — 计算给定 DEM 的坡度。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);

描述

计算给定 DEM 的坡度。返回坡度值。

units 指定返回值的单位。RADIANS, DEGREES(默认), PERCENT 等。

scale 指定栅格单元的大小。例如: 指定 scale=370400, 则: 指定 scale=111120 指定栅格单元的大小。

interpolate_nodata 指定, 指定 ST_InvDistWeight4ma 指定 NODATA 指定。



Note

(slope), (aspect), (hillshade) 指定, ESRI - How hillshade works 指定 ERDAS Field Guide - Slope Images 指定。

2.0.0 指定。

指定: 2.1.0 指定 ST_MapAlgebra() 指定, 指定 units, scale, interpolate_nodata 指定。

指定: 2.1.0 指定。2.1.0 指定。

指定: 指定 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},{21.568
{26.5650520324707,36.8698959350586,0,36.8698959350586,26.5650520324707},{21.5681285858154,35.2643890
5681285858154,26.5650520324707,21.5681285858154,10.0249881744385}}")
(1 row)
```

指定: 指定 2

Complete example of tiles of a coverage.

```

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 pixeltype="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 ☐☐☐☐☐☐☐☐☐☐.

例

```
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, 2, 3, 3],
      [1, 2, 4, 15, 3],
      [2, 4, 10, 7, 5],
      [3, 5, 7, 6, 4],
      [3, 3, 5, 4, 4]
    ]::double precision[][])
  ) AS rast
)
SELECT ST_Value(ST_TPI(rast, 1, '32BF'), 3, 3) AS tpi
FROM foo;
```

3.75

例

ST_MapAlgebra (callback function version), ST_TRI, ST_Roughness, ST_Slope, ST_HillShade, ST_Aspect

11.14.6 ST_TRI

ST_TRI — 地形崎岖度指数 (Terrain Ruggedness Index) 函数。

Synopsis

raster **ST_TRI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

例

地形崎岖度指数 (Terrain Ruggedness Index) 函数。



Note

默认 1 (focalmean radius of one) 函数。

2.1.0 版本。

例

```
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, 2, 3, 3],
```



```

        [1, 2, 4, 15, 3],
        [2, 4, 10, 7, 5],
        [3, 5, 7, 6, 4],
        [3, 3, 5, 4, 4]
    ]::double precision[][])
    ) AS rast
)
SELECT ST_Value(ST_TRI(rast, 1, '32BF'), 3, 3) AS tri
FROM foo;

```

5

☒☒

[ST_MapAlgebra \(callback function version\)](#), [ST_Roughness](#), [ST_TPI](#), [ST_Slope](#), [ST_HillShade](#), [ST_Aspect](#)

11.14.7 ST_InterpolateRaster

ST_InterpolateRaster — Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation.

Synopsis

raster **ST_InterpolateRaster**(geometry input_points, text algorithm_options, raster template, integer template_band_num=1);

☒☒

Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation. There are five interpolation algorithms available: inverse distance, inverse distance nearest-neighbor, moving average, nearest neighbor, and linear interpolation. See the [gdal_grid documentation](#) for more details on the algorithms and their parameters. For more information on how interpolations are calculated, see the [GDAL grid tutorial](#).

Input parameters are:

input_points The points to drive the interpolation. Any geometry with Z-values is acceptable, all points in the input will be used.

algorithm_options A string defining the algorithm and algorithm options, in the format used by [gdal_grid](#). For example, for an inverse-distance interpolation with a smoothing of 2, you would use "invdist:smoothing=2.0"

template A raster template to drive the geometry of the output raster. The width, height, pixel size, spatial extent and pixel type will be read from this template.

template_band_num By default the first band in the template raster is used to drive the output raster, but that can be adjusted with this parameter.

Availability: 3.2.0

❏

```
SELECT ST_InterpolateRaster('MULTIPOINT(10.5 9.5 1000, 11.5 8.5 1000, 10.5 8.5 500, 11.5 9.5 500)')::geometry,
      'invdist:smoothing:2.0',
      ST_AddBand(ST_MakeEmptyRaster(200, 400, 10, 10, 0.01, -0.005, 0, 0), '16BSI')
)
```

❏

ST_Contour

11.14.8 ST_Contour

ST_Contour — Generates a set of vector contours from the provided raster band, using the [GDAL contouring algorithm](#).

Synopsis

setof record **ST_Contour**(raster rast, integer bandnumber=1, double precision level_interval=100.0, double precision level_base=0.0, double precision[] fixed_levels=ARRAY[], boolean polygonize=false);

❏

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

例

```
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;
```

例

ST_InterpolateRaster

11.15 几何函数

11.15.1 Box3D

Box3D — 返回 BOX3D 几何体。

Synopsis

box3d **Box3D**(raster rast);

例

返回 BOX3D 几何体。

返回 ((MINX, MINY), (MAXX, MAXY)) 的 BOX3D 几何体。

版本: 2.0.0 返回 BOX3D 几何体 BOX2D 几何体。BOX2D 几何体, 2.0.0 返回 BOX3D 几何体。

例

```
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)

例

ST_Envelope

11.15.2 ST_ConvexHull

`ST_ConvexHull` — `BandNoDataValue` 返回的栅格值, 与 `ST_Envelope` 返回的几何值相同。 `ST_ConvexHull` 返回的几何值与 `ST_Envelope` 返回的几何值相同。

Synopsis

geometry **ST_ConvexHull**(raster rast);

`NoDataBandValue` 返回的栅格值, 与 `ST_Envelope` 返回的几何值相同。 `ST_ConvexHull` 返回的几何值与 `ST_Envelope` 返回的几何值相同。



Note

`ST_Envelope` 返回的几何值 (floor) 与 `ST_ConvexHull` 返回的几何值相同。 `ST_ConvexHull` 返回的几何值与 `ST_Envelope` 返回的几何值相同。

PostGIS Raster Specification 文档。

Note envelope and convexhull are more or less the same.

```
SELECT ST_AsText(ST_ConvexHull(rast)) As convexhull,
       ST_AsText(ST_Envelope(rast)) As env
FROM dummy_rast WHERE rid=1;
```

convexhull		env
-----+-----<		
POLYGON((0.5 0.5,20.5 0.5,20.5 60.5,0.5 60.5,0.5 0.5))		POLYGON((0 0,20 0,20 60,0 60,0 0))<

Now we skew the raster. Note how the convex hull and envelope are now different.

```
SELECT ST_AsText(ST_ConvexHull(rast)) As convexhull,
       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;
```

convexhull		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` — 将 `geomval(geom, val)` 返回的集合转换为多边形。返回的集合包含 1 个多边形。

Synopsis

setof geomval **ST_DumpAsPolygons**(raster rast, integer band_num=1, boolean exclude_nodata_value=TRUE)

返回

`ST_DumpAsPolygons` (SRF; Set-Returning Function) 返回。将 `(geom)` 返回的集合 `(val)` 转换为多边形 `geomval` 返回的集合。返回的集合包含 `val` 返回的集合/返回的集合。

`ST_DumpAsPolygon` 返回的集合。返回的集合包含 `GROUP BY` 返回的集合。

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

Changed 3.7.0, the polygonization honours PostgreSQL interrupts so cancellations and statement timeouts halt processing promptly.

GDAL 1.7 返回的集合。



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

返回的集合, `ST_ValueCount` 返回的集合。



Note

返回的集合 `ST_PixelAsPolygons` 返回的集合。

返回

This syntax uses a LATERAL join.

```
SELECT val, ST_AsText(geom) As geomwkt
FROM (
  SELECT dp.*
  FROM dummy_rast, LATERAL ST_DumpAsPolygons(rast) AS dp
  WHERE rid = 2
) As foo
WHERE val BETWEEN 249 and 251
ORDER BY val;
```

val	geomwkt
249	POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 5793243.85,3427928 5793243.95,3427927.95 5793243.95))
250	POLYGON((3427927.75 5793243.9,3427927.75 5793243.85,3427927.8 5793243.85,3427927.8 5793243.9,3427927.75 5793243.9))
250	POLYGON((3427927.8 5793243.8,3427927.8 5793243.75,3427927.85 5793243.75,3427927.85 5793243.8, 3427927.8 5793243.8))
251	POLYGON((3427927.75 5793243.85,3427927.75 5793243.8,3427927.8 5793243.8,3427927.8 5793243.85,3427927.75 5793243.85))

返回

[geomval](#), [ST_AsRasterAgg](#), [ST_Value](#), [ST_Polygon](#), [ST_ValueCount](#)

11.15.4 ST_Envelope

ST_Envelope — 返回给定栅格的边界框。

Synopsis

geometry **ST_Envelope**(raster rast);

返回

返回的几何体具有与输入栅格相同的 SRID。如果输入栅格的 SRID 为 0，则返回的几何体将具有 float8 精度。

返回的几何体将具有 ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY))。

返回

```
SELECT rid, ST_AsText(ST_Envelope(rast)) As envgeomwkt
FROM dummy_rast;
```

rid	envgeomwkt
1	POLYGON((0 0,20 0,20 60,0 60,0 0))
2	POLYGON((3427927 5793243,3427928 5793243,3427928 5793244,3427927 5793244, 3427927 5793243))

返回

[ST_Envelope](#), [ST_AsText](#), [ST_SRID](#)

11.15.5 ST_MinConvexHull

ST_MinConvexHull — 返回给定几何体的最小凸包。如果输入为 NULL，则返回 NULL。

Synopsis

geometry **ST_MinConvexHull**(raster rast, integer nband=NULL);

NOTES NODATA values are ignored. nband NULL, all bands are used.

2.1.0 Added.

```
WITH foo AS (
  SELECT
    ST_SetValues(ST_SetValues(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(9, 9, 0, 0, 1, ←
      -1, 0, 0, 0), 1, '8BUI', 0, 0), 2, '8BUI', 1, 0),
      1, 1, 1,
      ARRAY[
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0, 1],
        [0, 0, 0, 1, 1, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0]
      ]::double precision[][])
    ),
    2, 1, 1,
    ARRAY[
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [1, 0, 0, 0, 0, 1, 0, 0, 0],
      [0, 0, 0, 0, 1, 1, 0, 0, 0],
      [0, 0, 0, 0, 0, 1, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 1, 0, 0, 0, 0, 0, 0]
    ]::double precision[][])
  ) AS rast
)
SELECT
  ST_AsText(ST_ConvexHull(rast)) AS hull,
  ST_AsText(ST_MinConvexHull(rast)) AS mhull,
  ST_AsText(ST_MinConvexHull(rast, 1)) AS mhull_1,
  ST_AsText(ST_MinConvexHull(rast, 2)) AS mhull_2
FROM foo
```

hull		mhull		mhull_1	↔
		mhull_2			
-----+-----					
POLYGON((0 0,9 0,9 -9,0 -9,0 0))		POLYGON((0 -3,9 -3,9 -9,0 -9,0 -3))		POLYGON((3 -3,9	↔
-3,9 -6,3 -6,3 -3))		POLYGON((0 -3,6 -3,6 -9,0 -9,0 -3))			

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;
```

```
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**

11.15.7 ST_IntersectionFractions

ST_IntersectionFractions — Calculates the fraction of each raster cell that is covered by a given geometry.

Synopsis

raster **ST_IntersectionFractions**(raster rast, geometry geom);

☒☒

Calculates the fraction of each raster cell that is covered by a given geometry. The first argument is a raster, which defines the grid geometry to use for the calculation. The extent and cell size are read from the raster parameter. The second argument is a geometry, which is overlaid with the grid, and each grid populated based on overlaying the geometry on the grid. For polygons, the value returned for each cell is the proportion of its area that is covered by the geometry. For linestrings, the value returned for each cell is the length contained in the cell.

Availability: 3.6.0 Requires GEOS 3.14 or higher.

☒☒

```
CREATE TABLE raster_proportions_rast (
    name text,
    rast raster
);

INSERT INTO raster_proportions_rast (name, rast) VALUES (
    '2x2 raster covering 0,0 to 10,10',
    ST_MakeEmptyRaster(2, 2, -- raster width/height in pixels
        0, 10, -- upper-left corner x/y coordinates
        5, -5, -- pixel width/height in ground units
        0, 0, -- skew x/y
        0 -- SRID
```

```
));

--
-- This rotated square polygon covers half of each cell in the
-- raster.
--
SELECT name, ST_DumpValues(
    ST_IntersectionFractions(
        rast,
        'POLYGON((5 0, 0 5, 5 10, 10 5, 5 0))'::geometry),
        1)
FROM raster_proportions_rast;
```

```
2x2 raster covering 0,0 to 10,10
-----
{{0.5,0.5},{0.5,0.5}}
```

☒☒

ST_MakeEmptyRaster

11.16 空间函数

11.16.1 &&

&& — A 栅格与 B 栅格/几何体相交时返回 TRUE 或 FALSE。

Synopsis

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

☒☒

&& 栅格/几何体 A 与 栅格/几何体 B 相交时返回 TRUE 或 FALSE。



Note

栅格/几何体 (operand) 必须是有效的 PostGIS 几何体。

2.0.0 版本引入。

☒☒

```
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);

&< 返回 A 与 B 的交集是否为 TRUE，如果交集不为空，则返回 TRUE，否则返回 FALSE。



Note

（operand）必须为栅格类型。

```
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);

例

> 返回 A 与 B 的栅格交集, 返回 TRUE 的栅格。
 B 与 A 的栅格交集, TRUE 的栅格。



Note

运算符 (operand) 的返回值是 TRUE 或 FALSE。

例

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &
> B.rast As overright
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

a_rid	b_rid	overright
2	2	t
2	3	t
2	1	t
3	2	f
3	3	t
3	1	f
1	2	f
1	3	t
1	1	t

11.16.4 =

= — A 与 B 的栅格交集 TRUE 的栅格。返回 TRUE 的栅格。

Synopsis

boolean =(raster A , raster B);

例

= 返回 A 与 B 的栅格交集 TRUE 的栅格。 PostgreSQL 返回 TRUE 的栅格, 返回 FALSE 的栅格。 =, <, > 运算符 (GROUP BY ORDER BY)。



Caution

运算符 (operand) 的返回值是 TRUE 或 FALSE。 ~ = 运算符 (group by) 的返回值。

2.1.0 版本。

返回

~ =

11.16.5 @

@ — B 与 A 相交是否返回 TRUE。相交是指 B 与 A 的边界或内部重叠。

Synopsis

```
boolean @( raster A , raster B );
boolean @( geometry A , raster B );
boolean @( raster B , geometry A );
```

返回

@ 返回 B 与 A 相交是否返回 TRUE。



Note

相交 (operand) 是指 B 与 A 的边界或内部重叠。

2.0.0 返回 raster @ raster, raster @ geometry。

2.0.5 返回 geometry @ raster。

返回

~

11.16.6 ~=

~= — A 与 B 相交是否返回 TRUE。

Synopsis

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

返回

~= 返回 A 与 B 相交是否返回 TRUE。



Note

相交 (operand) 是指 B 与 A 的边界或内部重叠。

2.0.0 返回。

例

将精度为 2 的栅格数据与精度为 1 的栅格数据相加，并返回精度为 1 的结果。

```
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 的交集是否为 TRUE。如果交集不为空，则返回 TRUE。

Synopsis

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

例

~ 返回 TRUE 当且仅当 A 与 B 的交集不为空。



Note

~ 的 (operand) 必须是栅格或几何类型。

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 rastA 包含 rastB，则返回 TRUE。如果 rastA 包含 rastB，则返回 TRUE。

Synopsis

```
boolean ST_Contains( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Contains( raster rastA , raster rastB );
```

注意

`ST_Contains(rastA, rastB)` 返回 `TRUE` 当且仅当 `rastB` 完全包含在 `rastA` 中。如果 `rastA` 或 `rastB` 包含 `NULL` 值，则返回 `NULL`。如果 `rastA` 或 `rastB` 包含 `NODATA` 值，则返回 `FALSE`。

Note!

Note

`ST_Contains(rastA, rastB)` 返回 `TRUE` 当且仅当 `rastB` 完全包含在 `rastA` 中。

Note!

Note

`ST_Contains(rastA, rastB)` 返回 `TRUE` 当且仅当 `rastB` 完全包含在 `rastA` 中。如果 `rastA` 或 `rastB` 包含 `NULL` 值，则返回 `NULL`。如果 `rastA` 或 `rastB` 包含 `NODATA` 值，则返回 `FALSE`。

Note!

Note

`ST_Contains()` 和 `ST_Within()` 返回 `TRUE` 当且仅当 `ST_Contains(rastA, rastB)` 返回 `TRUE`。如果 `rastA` 或 `rastB` 包含 `NULL` 值，则返回 `NULL`。如果 `rastA` 或 `rastB` 包含 `NODATA` 值，则返回 `FALSE`。

2.1.0 版本中，`ST_Contains()` 返回 `TRUE`。

注意

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	t
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

注意

[ST_Intersects](#), [ST_Within](#)

11.17.2 ST_ContainsProperly

ST_ContainsProperly — rastB 是否完全包含在 rastA 中且 rastA 与 rastB 不相交。

Synopsis

```
boolean ST_ContainsProperly( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_ContainsProperly( raster rastA , raster rastB );
```

返回

如果 rastB 完全包含在 rastA 中且 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 包含 NODATA 值，则返回 false。

如果 rastA 与 rastB 不相交，则返回 false。



Note

如果 rastA 与 rastB 不相交，则返回 false。



Note

如果 rastA 与 rastB 不相交，则返回 false。ST_ContainsProperly(ST_Polygon(raster), geometry) 与 ST_ContainsProperly(geometry, ST_Polygon(raster)) 返回 true 当且仅当 ST_Polygon 完全包含在 geometry 中。

2.1.0 版本引入。

返回

```
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
```

返回

ST_Intersects, ST_Contains

11.17.3 ST_Covers

ST_Covers — 是否 rastB 完全覆盖 rastA。

Synopsis

```
boolean ST_Covers( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Covers( raster rastA , raster rastB );
```

返回

如果 rastB 完全包含 rastA 的所有像素，则 ST_Covers(rastA, rastB) 返回 true。如果 rastB 包含 rastA 的所有像素，但 rastA 包含一些 NULL 像素，则 ST_Covers(rastA, rastB) 返回 false。如果 rastB 包含 rastA 的所有像素，但 rastA 包含一些 NOData 像素，则 ST_Covers(rastA, rastB) 返回 false。



Note

ST_Covers 与 ST_Contains 类似，但 ST_Covers 考虑了像素的覆盖关系。



Note

ST_Covers(ST_Polygon(raster), geometry) 与 ST_Covers(geometry, ST_Polygon(raster)) 返回相同的布尔值。

2.1.0 版本引入。

返回

```
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
```

返回

ST_Intersects, **ST_CoveredBy**

11.17.4 ST_CoveredBy

ST_CoveredBy — 如果 rastA 完全包含 rastB 的所有像素，则 ST_CoveredBy(rastA, rastB) 返回 true。

Synopsis

```
boolean ST_CoveredBy( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_CoveredBy( raster rastA , raster rastB );
```




Note

ST_Difference(raster, geometry) 返回一个多边形。



Note

ST_Difference(raster, geometry) 和 ST_Disjoint(ST_Polygon(raster), geometry) 返回一个布尔值。ST_Polygon 返回一个多边形。

2.1.0 简体中文。

SQL

Rid = 1 has no bands, hence the NOTICE and the NULL value for st_disjoint.

```
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN  
dummy_rast r2 WHERE r1.rid = 2;
```

NOTICE: The second raster provided has no bands

rid	rid	st_disjoint
2	1	
2	2	f

This time, without specifying band numbers.

```
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN  
dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_disjoint
2	1	t
2	2	f

SQL

ST_Intersects

11.17.6 ST_Intersects

ST_Intersects — 返回 rasterA 和 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 );
```


**Note**

ST_Overlaps 与 ST_Contains 不同，ST_Overlaps 要求两个多边形至少有一个公共点，且该点不在边界上。

**Note**

ST_Overlaps(ST_Polygon(raster), geometry) 与 ST_Polygon 返回的几何体不同。

2.1.0 版本引入。

比较

Comparing different bands of same raster.

```
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;
```

f

比较

ST_Intersects

11.17.8 ST_Touches

ST_Touches — 如果 rastA 与 rastB 至少有一个公共点，且该点不在边界上，则返回 TRUE。

Synopsis

boolean **ST_Touches**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Touches**(raster rastA , raster rastB);

比较

如果 rastA 与 rastB 至少有一个公共点，且该点不在边界上，则返回 TRUE。如果 rastA 与 rastB 没有公共点，则返回 FALSE。如果 rastA 或 rastB 为 NULL，则返回 NULL。如果 rastA 或 rastB 为 (NODATA 值)，则返回 NULL。

**Note**

ST_Touches 与 ST_Contains 不同，ST_Touches 要求两个多边形至少有一个公共点，且该点不在边界上。

**Note**

ST_Touches(ST_Polygon(raster), geometry) 与 ST_Polygon 返回的几何体不同。

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)。如果两个栅格的单元大小、偏移量、旋转角度、SRID 和投影类型都相同，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

例 1, 2: 检查两个栅格是否具有相同的对齐方式。第一个栅格的单元大小为 10 米，SRID 为 4326。第二个栅格的单元大小为 20 米，SRID 为 4326。由于单元大小不同，返回 false。 (NOTICE) 检查单元大小。

例 3: 检查两个栅格是否具有相同的对齐方式。使用 ST_SameAlignment 函数。在 PostgreSQL 中，该函数是一个聚合函数。使用 SUM() 或 AVG() 函数可以检查多个栅格是否具有相同的对齐方式。

2.0.0 版本引入。

更新: 2.1.0 版本引入。

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;
```

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
t
f
f
f
```

文档

Section 10.1, [ST_NotSameAlignmentReason](#), [ST_MakeEmptyRaster](#)

11.17.10 ST_NotSameAlignmentReason

`ST_NotSameAlignmentReason` — 检查两个栅格是否具有相同的对齐方式，并返回一个包含原因代码的文本字符串。

Synopsis

```
text ST_NotSameAlignmentReason(raster rastA, raster rastB);
```

文档

返回一个文本字符串，其中包含一个或多个原因代码，这些代码由以下常量组成。



Note

如果两个栅格具有相同的对齐方式，则返回空字符串。如果返回非空字符串，则字符串中的每个字符都是一个原因代码（由以下常量组成）。

2.1.0 版本引入。

文档

```
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_notsamealignmentreason
f	The rasters have different scales on the X axis
(1 row)	

¶¶

Section 10.1, [ST_SameAlignment](#)

11.17.11 ST_Within

`ST_Within` — 检查 `rastB` 是否完全包含在 `rastA` 中, 检查 `rastA` 是否完全包含在 `rastB` 中。

Synopsis

boolean **ST_Within**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Within**(raster rastA , raster rastB);

¶¶

检查 `rastB` 是否完全包含在 `rastA` 中, 检查 `rastA` 是否完全包含在 `rastB` 中。
检查 `rastA` 是否包含在 `rastB` 中。如果 `rastA` 或 `rastB` 包含 NULL 值, 则返回 NULL。
如果 `rastA` 或 `rastB` 包含 (NODATA 值) 则返回 NULL。



Note
检查 (operand) 是否完全包含在检查中。



Note
检查 `ST_Within(ST_Polygon(raster), geometry)` 是否等于 `ST_Within(geometry, ST_Polygon(raster))` 是否等于 `ST_Polygon` 是否完全包含在检查中。



Note
`ST_Within()` 与 `ST_Contains()` 相反。检查, `ST_Within(rastA, rastB)` 是否等于 `ST_Contains(rastB, rastA)` 是否等于检查。

2.1.0 检查是否完全包含在检查中。

¶¶

```
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 当且仅当 rastA 完全在 rastB 内部，且 rastA 与 rastB 之间的距离大于 distance_of_srid。

如果 distance_of_srid 是负数，则返回 TRUE 当且仅当 rastA 完全在 rastB 内部，且 rastA 与 rastB 之间的距离小于 distance_of_srid。如果 distance_of_srid 是 0，则返回 FALSE。



Note

如果 distance_of_srid 是 0，则返回 FALSE。



Note

ST_DFullyWithin(ST_Polygon(raster), geometry) 与 ST_DFullyWithin(ST_Polygon(raster), geometry) 相同。

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.

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_ClusterRelateWin** - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.
 - **ST_ClusterWithin** - Aggregate function that clusters geometries by separation distance.
 - **ST_ClusterWithinWin** - Window function that returns a cluster id for each input geometry, clustering using separation distance.
 - **ST_Collect** - Creates a GeometryCollection or Multi* geometry from a set of geometries.
 - **ST_CoverageClean** - Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.
 - **ST_CoverageInvalidEdges** - Window function that finds locations where polygons fail to form a valid coverage.
-

- **ST_CoverageSimplify** - Window function that simplifies the edges of a polygonal coverage.
- **ST_CoverageUnion** - Computes the union of a set of polygons forming a coverage by removing shared edges.
- **ST_Extent** - Aggregate function that returns the bounding box of geometries.
- **ST_MakeLine** - 返回由给定的点集组成的线。如果点集是空的，则返回空几何。
- **ST_MemUnion** - Aggregate function which unions geometries in a memory-efficient but slower way.
- **ST_MinimumSpanningTree** - Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.
- **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。如果两个几何体具有相同的对齐方式，则返回 true；否则返回 false。
- **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。如果点集是空的，则返回空几何。
- **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_ClusterRelateWin** - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.
- **ST_ClusterWithinWin** - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- **ST_CoverageClean** - Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.
- **ST_CoverageInvalidEdges** - Window function that finds locations where polygons fail to form a valid coverage.
- **ST_CoverageSimplify** - Window function that simplifies the edges of a polygonal coverage.
- **ST_MinimumSpanningTree** - Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.

13.3 PostGIS SQL-MM Compliant Functions

The functions given below are PostGIS functions that conform to the SQL/MM 3 standard

- **CG_3DArea** - 3 维几何体的面积。返回 0 或正数。
- **CG_3DDifference** - 3 维几何体的差集。
- **CG_3DIntersection** - 3 维几何体的交集。
- **CG_3DUnion** - Perform 3D union using postgis_sfcgal.
- **CG_Volume** - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **ST_3DArea** - 3 维几何体的面积。返回 0 或正数。
- **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** - 返回两个 3D 几何体 (SRS 相同) 之间的最短距离。
- **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_3DLength** - 返回 3D 线段的长度。
- **ST_3DPerimeter** - 返回 3D 多边形的周长。
- **ST_3DUnion** - Perform 3D union.
- **ST_AddEdgeModFace** - 添加边并修改面，返回修改后的面。
- **ST_AddEdgeNewFaces** - 添加边并创建新面，返回新面。
- **ST_AddIsoEdge** - 添加孤立边，返回边 ID。
- **ST_AddIsoNode** - 添加孤立节点，返回节点 ID。
- **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 格式的几何体。
- **ST_AsText** - 返回 WKT(Well-Known Text) 格式的几何体。
- **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 返回其坐标维数。
- **ST_CreateTopoGeo** - 从几何集合创建拓扑几何。返回的拓扑几何包含所有输入几何的拓扑关系。
- **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 返回其维数。
- **ST_Disjoint** - Tests if two geometries have no points in common
- **ST_Distance** - 返回两个几何体之间的最短距离 (longest) 距离。
- **ST_EndPoint** - Returns the last point of a LineString, CircularLineString, or NURBSCurve.
- **ST_Envelope** - 返回几何体的包络面 (double precision; float8) 几何体。
- **ST_Equals** - Tests if two geometries include the same set of points
- **ST_ExteriorRing** - 返回几何体的外部环。
- **ST_GMLToSQL** - GML 几何体转换为 ST_Geometry。使用 ST_GeomFromGML 函数。
- **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 几何体转换为 ST_Geometry。
- **ST_GeomFromWKB** - WKB(Well-Known Binary) 几何体转换为 SRID 几何体。
- **ST_GeometryFromText** - WKT(Well-Known Text) 几何体转换为 ST_Geometry。使用 ST_GeomFromText 函数。
- **ST_GeometryN** - ST_Geometry 返回其第 N 个几何体。
- **ST_GeometryType** - ST_Geometry 返回其几何体类型。
- **ST_GetFaceEdges** - aface 返回其面边缘。
- **ST_GetFaceGeometry** - 返回其面 ID 的几何体。
- **ST_InitTopoGeo** - Creates a new topology schema and registers it in the topology.topology table.
- **ST_InteriorRingN** - 返回其第 N 个内部环。
- **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。对于其他几何体类型，返回 FALSE。
- **ST_IsEmpty** - Tests if a geometry is empty.
- **ST_IsRing** - Tests if a LineString is closed and simple.

- **ST_IsSimple** - 测试几何体是否是简单的。如果几何体是简单的，返回 TRUE，否则返回 FALSE。
- **ST_IsValid** - Tests if a geometry is well-formed in 2D.
- **ST_Length** - 返回几何体的长度。
- **ST_LineFromText** - 从 WKT 字符串创建 Geometry。SRID 默认为 0。
- **ST_LineFromWKB** - 从 WKB 二进制数据创建 Geometry。
- **ST_LinestringFromWKB** - 从 WKB 二进制数据创建 LineString。
- **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 字符串创建 MultiLineString。
- **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** - 分割边并返回新的边 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** - 分割边并返回新的边 ID。
- **ST_NumCurves** - Return the number of component curves in a CompoundCurve.
- **ST_NumGeometries** - 返回几何体数组中的几何体数量。
- **ST_NumInteriorRings** - 返回多边形中的内部环数量。
- **ST_NumPatches** - 返回几何体中的补丁数量。如果几何体是 NULL，则返回 NULL。
- **ST_NumPoints** - ST_LineString 或 ST_CircularString 中的点数。
- **ST_OrderingEquals** - Tests if two geometries represent the same geometry and have points in the same directional order
- **ST_Overlaps** - Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other
- **ST_PatchN** - ST_Geometry 数组中的第 N 个补丁。
- **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** - 使用 SRID 和 WKT 创建点几何体。SRID 默认为 0。
 - **ST_PointFromWKB** - 使用 SRID 和 WKB 创建点几何体。
 - **ST_PointN** - 从 ST_LineString 或 ST_CircularString 中获取第 N 个点。
 - **ST_PointOnSurface** - 计算一个点，保证该点位于多边形内，或在几何体上。
 - **ST_Polygon** - 从 LineString 创建多边形，指定 SRID。
 - **ST_PolygonFromText** - 从 WKT 创建几何体，指定 SRID。如果 SRID 未指定，默认为 0。
 - **ST_Relate** - 测试两个几何体是否具有匹配的拓扑关系模式，或计算它们的交集矩阵。
 - **ST_RemEdgeModFace** - 移除一条边，如果该边分隔两个面，则删除一个面并修改另一个面以覆盖其空间。
 - **ST_RemEdgeNewFace** - 移除一条边，并创建新的面。如果该边分隔两个面，则删除一个面并创建新的面以覆盖其空间。
 - **ST_RemoveIsoEdge** - 移除一条孤立的边并返回描述。如果边不是孤立的，则抛出异常。
 - **ST_RemoveIsoNode** - 移除一个孤立的节点并返回描述。如果节点不是孤立的，则抛出异常。
 - **ST_SRID** - 返回几何体的空间参考标识符。
 - **ST_StartPoint** - 返回 LineString、CircularLineString 或 NURBSCurve 的第一个点。
 - **ST_SymDifference** - 计算两个几何体 A 和 B 的对称差集。
 - **ST_Touches** - 测试两个几何体是否至少有一个公共点，但它们的内部不相交。
 - **ST_Transform** - 返回一个具有坐标转换到新空间参考系统的几何体。
 - **ST_Union** - 计算两个几何体 A 和 B 的并集。
 - **ST_Volume** - 计算 3D 几何体的体积。封闭曲面可以有非零体积。
 - **ST_WKBToSQL** - 将 WKB(Well-Known Binary) 格式转换为 ST_Geometry 格式。指定 SRID 默认为 0。
 - **ST_WKTToSQL** - 将 WKT(Well-Known Text) 格式转换为 ST_Geometry 格式。指定 SRID 默认为 0。
 - **ST_Within** - 测试几何体 A 是否完全位于几何体 B 内，且它们的内部有公共点。
 - **ST_X** - 返回点的 X 坐标。
 - **ST_Y** - 返回点的 Y 坐标。
 - **ST_Z** - 返回点的 Z 坐标。
 - **ST_SRID** - 返回拓扑几何体的空间参考标识符。
-

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** - 返回几何图形的面积。
- **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) 格式的 SRID 几何图形。
- **ST_AsGML** - 返回 GML 2 或 GML 3 格式的几何图形。
- **ST_AsGeoJSON** - Return a geometry or feature in GeoJSON format.
- **ST_AsKML** - 返回 GML 2 或 GML 3 格式的几何图形。
- **ST_AsSVG** - Returns SVG path data for a geometry.
- **ST_AsText** - 返回 WKT(Well-Known Text) 格式的 SRID 几何图形。
- **ST_Azimuth** - 返回从第一个几何图形到第二个几何图形的方位角。
- **ST_Buffer** - Computes a geometry covering all points within a given distance from a geometry.
- **ST_Centroid** - 返回几何图形的质心。
- **ST_ClosestPoint** - Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.
- **ST_CoveredBy** - Tests if every point of A lies in B
- **ST_Covers** - Tests if every point of B lies in A
- **ST_DWithin** - Tests if two geometries are within a given distance
- **ST_Distance** - 返回两个几何图形之间的最短距离 (longest)。
- **ST_GeogFromText** - WKT (几何) 转换为 geography 类型。
- **ST_GeogFromWKB** - WKB 转换为 EWKB(几何 WKB) 格式。
- **ST_GeographyFromText** - WKT (几何) 转换为 geography 类型。
- **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **ST_Intersection** - Computes a geometry representing the shared portion of geometries A and B.
- **ST_Intersects** - Tests if two geometries intersect (they have at least one point in common)
- **ST_Length** - 返回几何图形的长度。
- **ST_LineInterpolatePoint** - Returns a point interpolated along a line at a fractional location.

- ## 13.5 PostGIS Raster Support Functions

- **Box3D** - `Box3D` `Box3D`.
- **@** - `B` `A` `TRUE` `TRUE`. `TRUE`.
- **~** - `A` `B` `TRUE` `TRUE`. `TRUE`.
- **=** - `A` `B` `TRUE` `TRUE`. `TRUE`.
- **&&** - `A` `B` `TRUE` `TRUE`.
- **&<** - `A` `B` `TRUE` `TRUE`.
- **&>** - `A` `B` `TRUE` `TRUE`.
- **~=** - `A` `B` `TRUE` `TRUE`.
- **ST_Retile** - `ST_Retile`, `ST_Retile`.
- **ST_AddBand** - `ST_AddBand` (`ST`) `ST`. `ST`, `ST`.
- **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** - `ST_AsJPEG` JPEG(Joint Photographic Exports Group) `ST` (`ST`) `ST`. `ST` 1 `ST` 3 `ST` 3 `ST` 3 `ST` RGB `ST`.
- **ST_AsPNG** - `ST_AsPNG` PNG(Portable Network Graphics) `ST` (`ST`) `ST`. `ST` 1 `ST`, 3 `ST`, `ST` 4 `ST` `ST`. `ST` 2 `ST` 4 `ST` `ST`. `ST` 1 `ST`. `ST` RGB `ST` RGBA `ST`.

- **ST_AsRaster** - PostGIS 将 PostGIS 几何体转换为栅格。
- **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** - 计算栅格的坡度 (度) 并返回。返回值的范围是 0 到 90。
- **ST_Band** - 返回指定 bandnum 的栅格元数据。如果 bandnum 未指定，则返回 1 的元数据。
- **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** - 检查 NODATA 值是否存在于指定 bandnum 的栅格中。
- **ST_BandMetaData** - 返回指定 bandnum 的栅格元数据。如果 bandnum 未指定，则返回 1 的元数据。
- **ST_BandNoDataValue** - 返回指定 bandnum 的 NODATA 值。如果 bandnum 未指定，则返回 1 的 NODATA 值。
- **ST_BandPath** - 返回指定 bandnum 的栅格文件路径。bandnum 未指定时返回 1 的路径。
- **ST_BandPixelType** - 返回指定 bandnum 的栅格像素类型。bandnum 未指定时返回 1 的像素类型。
- **ST_Clip** - Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.
- **ST_ColorMap** - 将 8BUI 栅格 (grayscale, RGB, RGBA) 转换为 4 字节的栅格。如果 bandnum 未指定，则返回 1 的栅格。
- **ST_Contains** - 检查 rastA 是否包含 rastB。如果 rastB 包含 rastA，则返回 TRUE。
- **ST_ContainsProperly** - rastB 是否完全包含在 rastA 中。如果 rastB 完全包含在 rastA 中，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 指定 NODATA 值，ST_Envelope 指定返回的栅格范围。如果 bandnum 未指定，则返回 1 的栅格。
- **ST_Count** - 计算指定 bandnum 的栅格中像素的数量。exclude_nodata_value 指定是否排除 NODATA 值。如果 bandnum 未指定，则返回 1 的计数。
- **ST_CountAgg** - 聚合函数，计算指定 bandnum 的栅格中像素的数量。exclude_nodata_value 指定是否排除 NODATA 值。如果 bandnum 未指定，则返回 1 的计数。
- **ST_CoveredBy** - 检查 rastA 是否被 rastB 覆盖。如果 rastB 覆盖 rastA，则返回 TRUE。
- **ST_Covers** - 检查 rastB 是否覆盖 rastA。如果 rastB 覆盖 rastA，则返回 TRUE。
- **ST_DFullyWithin** - 检查 rastA 是否完全在 rastB 的范围内。如果 rastA 完全在 rastB 的范围内，则返回 TRUE。

- **ST_DWithin** - 返回 rastA 和 rastB 在指定距离内的栅格单元。
- **ST_Disjoint** - 返回 rastA 和 rastB 是否不相交。
- **ST_DumpAsPolygons** - 将 geomval(geom, val) 转换为多边形。返回的数组包含 1 个元素。
- **ST_DumpValues** - 返回栅格单元 2 个值的数组。
- **ST_Envelope** - 返回栅格的包络线。
- **ST_FromGDALRaster** - 从 GDAL 栅格创建 PostGIS 栅格。
- **ST_GeoReference** - 将 (world) 坐标与 GDAL 的 ESRI 投影进行关联。返回 GDAL 的 SRS。
- **ST_Grayscale** - Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue
- **ST_HasNoBand** - 返回栅格是否具有指定的波段。返回 1 表示没有，0 表示有。
- **ST_Height** - 返回栅格的高度。
- **ST_HillShade** - 根据栅格值，使用指定的参数生成地形阴影图。
- **ST_Histogram** - 返回 (bin; 频率) 的数组，表示栅格值的分布。
- **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。
- **ST_IntersectionFractions** - Calculates the fraction of each raster cell that is covered by a given geometry.
- **ST_Intersects** - 返回 rastA 和 rastB 是否相交。
- **ST_IsEmpty** - 返回栅格是否为空 (width = 0, height = 0)。
- **ST_MakeEmptyCoverage** - Cover georeferenced area with a grid of empty raster tiles.
- **ST_MakeEmptyRaster** - 创建空栅格 (width & height), 指定 X 和 Y 坐标, 指定 SRS (scalex, scaley, skewx & skewy) 和 SRID (SRID) 的栅格。返回的栅格具有指定的 SRID。SRID 为 0 表示未知。
- **ST_MapAlgebra (callback function version)** - 返回 1 个栅格，表示两个栅格的代数运算结果。
- **ST_MapAlgebraExpr** - 返回 1 个栅格：使用 PostgreSQL 的表达式对两个栅格进行代数运算。
- **ST_MapAlgebraExpr** - 返回 2 个栅格：使用 PostgreSQL 的表达式对两个栅格进行代数运算。

- **ST_MapAlgebraFct** - 返回 1 个值: 使用 PostgreSQL 函数在栅格上执行代数操作。返回 1 个值。
- **ST_MapAlgebraFct** - 返回 2 个值: 使用 PostgreSQL 函数在栅格上执行代数操作。返回 2 个值。
- **ST_MapAlgebraFctNgb** - 返回 1 个值: 使用 PostgreSQL 函数在栅格上执行代数操作 (Map Algebra Nearest Neighbor)。返回 1 个值。
- **ST_MapAlgebra (expression version)** - 返回 1 个值: 使用 PostgreSQL 函数在栅格上执行代数操作 (expression version)。返回 1 个值。
- **ST_MemSize** - 返回栅格的内存大小 (字节)。
- **ST_MetaData** - 返回栅格的元数据, 包括 (skew), 等等。
- **ST_MinConvexHull** - 返回 NODATA 值。
- **ST_NearestValue** - columnx 和 rowy, 返回栅格中最近的值。返回 NODATA 值。
- **ST_Neighborhood** - columnx 和 rowy, 返回栅格中的邻域。返回 NODATA 值。
- **ST_NotSameAlignmentReason** - 返回栅格的不对齐原因。
- **ST_NumBands** - 返回栅格的波段数。
- **ST_Overlaps** - 返回 rastA 和 rastB 是否重叠。
- **ST_PixelAsCentroid** - 返回栅格的像素质心 (点)。
- **ST_PixelAsCentroids** - 返回栅格的像素质心 (点) 的集合 X, Y。
- **ST_PixelAsPoint** - 返回栅格的像素点。
- **ST_PixelAsPoints** - 返回栅格的像素点 X, Y 的集合。
- **ST_PixelAsPolygon** - 返回栅格的像素多边形。
- **ST_PixelAsPolygons** - 返回栅格的像素多边形 X, Y 的集合。
- **ST_PixelHeight** - 返回栅格的像素高度。
- **ST_PixelOfValue** - 返回栅格中指定值的像素 columnx, rowy。
- **ST_PixelWidth** - 返回栅格的像素宽度。
- **ST_Polygon** - NODATA 值。
- **ST_Quantile** - 返回栅格的 (population) 分位数 (quantile) 值。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** - $\text{ST\_RasterToWorldCoord}(x, y)$ returns the coordinates of the pixel at x, y in the raster.
- ST_RasterToWorldCoordX** - $\text{ST\_RasterToWorldCoordX}(x)$ returns the X coordinate of the pixel at x in the raster.
- ST_RasterToWorldCoordY** - $\text{ST\_RasterToWorldCoordY}(y)$ returns the Y coordinate of the pixel at y in the raster.
- ST_Reclass** - $\text{ST\_Reclass}(raster, nband, new\_values)$ reclassifies the raster. $nband$ is the number of bands. new_values is a table of values to reclassify to. Default is 16BUI or 8BUI.
- ST_ReclassExact** - Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.
- ST_Resample** - $\text{ST\_Resample}(raster, width, height, method)$ resamples the raster. $method$ can be NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos, Max or Min.
- 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** - $\text{ST\_Resize}(raster, width, height)$ resizes the raster.
- ST_Reskew** - $\text{ST\_Reskew}(raster, method)$ reskews the raster. $method$ can be NearestNeighbor, Bilinear, Cubic, CubicSpline or Lanczos.
- ST_Rotation** - $\text{ST\_Rotation}(raster, angle)$ rotates the raster.
- ST_Roughness** - DEM roughness (roughness) is a measure of the roughness of the terrain.
- ST_SRID** - $\text{ST\_SRID}(raster, srid)$ sets the SRID of the raster.
- ST_SameAlignment** - $\text{ST\_SameAlignment}(raster1, raster2)$ checks if two rasters have the same alignment.
- ST_ScaleX** - $\text{ST\_ScaleX}(raster, scale)$ scales the X coordinate of the raster.
- ST_ScaleY** - $\text{ST\_ScaleY}(raster, scale)$ scales the Y coordinate of the raster.
- ST_SetBandIndex** - Update the external band number of an out-db band.
- ST_SetBandIsNoData** - $\text{ST\_SetBandIsNoData}(raster, band, isnodata)$ sets the isnodata flag for a band.
- ST_SetBandNoDataValue** - $\text{ST\_SetBandNoDataValue}(raster, band, nodata\_value)$ sets the nodata value for a band.
- ST_SetBandPath** - Update the external path and band number of an out-db band.
- ST_SetGeoReference** - $\text{ST\_SetGeoReference}(raster, geo\_info)$ sets the geo reference information.
- 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** - $\text{ST\_SetRotation}(raster, angle)$ sets the rotation of the raster.
- ST_SetSRID** - $\text{ST\_SetSRID}(raster, srid)$ sets the SRID of the raster.
- ST_SetScale** - $\text{ST\_SetScale}(raster, x\_scale, y\_scale)$ sets the scale of the raster.

- **ST_SetSkew** - 设置 X 和 Y 的 skew (倾斜) 值。X 和 Y 的 skew 值默认为 0。
- **ST_SetUpperLeft** - Sets the value of the upper left corner of the pixel of the raster to projected X and Y coordinates.
- **ST_SetValue** - 设置 columnx, rowy 位置的像素值。如果 columnx 或 rowy 为 1，则表示第一列或第一行。
- **ST_SetValues** - 设置指定范围的像素值。
- **ST_SetZ** - Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimension using the requested resample algorithm.
- **ST_SkewX** - 设置 X 的 skew (倾斜) 值。
- **ST_SkewY** - 设置 Y 的 skew (倾斜) 值。
- **ST_Slope** - 计算指定范围的坡度 (斜率)。
- **ST_SnapToGrid** - 将栅格数据 snap 到指定的栅格。NearestNeighbor (最近邻), Bilinear, Cubic, CubicSpline 和 Lanczos 是支持的插值方法。NearestNeighbor 是默认方法。
- **ST_Summary** - 返回栅格统计信息。
- **ST_SummaryStats** - Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed if no band is specified.
- **ST_SummaryStatsAgg** - Aggregate. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a set of raster. Band 1 is assumed if no band is specified.
- **ST_TPI** - 计算地形位置指数 (Topographic Position Index)。
- **ST_TRI** - 计算地形崎岖度指数 (Terrain Ruggedness Index)。
- **ST_Tile** - 将栅格数据分割成指定的 tile 大小。
- **ST_Touches** - 检查两个栅格 (rastA 和 rastB) 是否接触。TRUE 表示接触。
- **ST_Transform** - 将栅格数据从一种坐标系转换到另一种坐标系。支持的插值方法有 NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos 和 NearestNeighbor。NearestNeighbor 是默认方法。
- **ST_Union** - 将两个栅格 (rastA 和 rastB) 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 columnx, rowy 位置的像素值。如果 columnx 或 rowy 为 1，则表示第一列或第一行。exclude_nodata_value 参数用于指定是否排除 nodata 值。
- **ST_ValueCount** - 返回指定范围内的像素值及其计数。如果 count 为 1，则表示只有一个像素值。NODATA 表示没有数据。
- **ST_Width** - 返回栅格的宽度。
- **ST_Within** - 检查栅格 B 是否在栅格 A 的范围内。TRUE 表示在范围内。

- **ST_WorldToRasterCoord** - 返回 X, Y(点, 点) 世界坐标到栅格坐标的转换。
- **ST_WorldToRasterCoordX** - 返回 (pt) 世界坐标到栅格坐标的转换 X, Y 栅格坐标 (xw, yw) 栅格坐标。
- **ST_WorldToRasterCoordY** - 返回 (pt) 世界坐标到栅格坐标的转换 X, Y 栅格坐标 (xw, yw) 栅格坐标。
- **UpdateRasterSRID** - 更新栅格 SRID 值。

13.6 PostGIS Geometry / Geography / Raster Dump Functions

The functions given below are PostGIS functions that take as input or return as output a set of or single **geometry_dump** or **geomval** data type object.

- **ST_DumpAsPolygons** - 返回 geomval(geom, val) 栅格坐标到栅格坐标的转换 1 栅格坐标。
- **ST_Intersection** - 返回两个栅格坐标的交集。
- **ST_Dump** - Returns a set of geometry_dump rows for the components of a geometry.
- **ST_DumpPoints** - 返回栅格坐标的栅格坐标。
- **ST_DumpRings** - Returns a set of geometry_dump rows for the exterior and interior rings of a Polygon.
- **ST_DumpSegments** - 返回栅格坐标的栅格坐标。

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 BOX3D 栅格坐标。
- **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) 栅格坐标。
- **ST_Box2dFromGeoHash** - GeoHash 栅格坐标 BOX2D 栅格坐标。
- **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_XSize** - Returns the X size 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_YSize** - Returns the Y size of a 2D or 3D bounding box or a geometry.
 - **ST_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
 - **ST_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
 - **ST_ZSize** - Returns the Z size of a 2D or 3D bounding box or a geometry.
 - **RemoveUnusedPrimitives** - Removes topology primitives which not needed to define existing Topo-Geometry objects.
 - **ValidateTopology** - Returns a set of validate_topology_returntype objects detailing issues with topology.
 - **ValidateTopologyPrecision** - Returns non-precise vertices in the topology.
 - **~(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
 - **~(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
 - **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (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** - 添加几何列。
- **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 维面积。返回 0 或正数。
- **CG_3DConvexHull** - 3 维凸包。
- **CG_3DDifference** - 3 维差集。
- **CG_3DIntersection** - 3 维交集。
- **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_ConstrainedDelaunayTriangles** - Return a constrained Delaunay triangulation around the given input geometry.
- **CG_Extrude** - 3 维拉伸。
- **CG_ForceLHR** - LHR(Left Hand Reverse; 左手反向) 强制。
- **CG_IsPlanar** - 是否是平面。
- **CG_IsSolid** - 是否是体。返回 0 或正数。
- **CG_MakeSolid** - 3 维体。返回 0 或正数。返回 0 表示不是体，返回正数表示是体。TIN 体。
- **CG_NurbsCurveApproximate** - Creates an approximating NURBS curve fitting data points within a tolerance
- **CG_NurbsCurveDerivative** - Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter
- **CG_NurbsCurveInterpolate** - Creates an interpolating NURBS curve passing through all given data points
- **CG_Orientation** - 3 维方位 (orientation) 返回。
- **CG_RotateX** - Rotates a geometry around the X-axis by a given angle.
- **CG_RotateY** - Rotates a geometry around the Y-axis by a given angle.
- **CG_RotateZ** - Rotates a geometry around the Z-axis by a given angle.
- **CG_Simplify** - Reduces the complexity of a geometry while preserving essential features and Z/M values.
- **CG_StraightSkeleton** - 3 维直线骨架 (straight skeleton) 返回。
- **CG_Tessellate** - 3 维三角化 (tessellation) 返回 TIN 体 TIN 体。

- **CG_Visibility** - Compute a visibility polygon from a point or a segment in a polygon geometry
- **CG_Volume** - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **DropGeometryColumn** - 删除指定表中的指定几何列。
- **GeometryType** - ST_Geometry 返回几何类型。
- **ST_3DArea** - 3D 几何体的面积。返回 0 表示空几何体。
- **ST_3DClosestPoint** - g2 与 g1 的 3D 最近点。返回 3D 最近点的几何体。
- **ST_3DConvexHull** - 3D 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** - 3D 几何体的差集。
- **ST_3DDistance** - 3D 几何体之间的距离 (SRS 无关) 3D 几何体之间的距离。
- **ST_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST_3DInterpolatePoint** - Returns the interpolated measure of a geometry closest to a point in 3D.
- **ST_3DIntersection** - 3D 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_3DLength** - 3D 几何体的长度。
- **ST_3DLineInterpolatePoint** - Returns a point interpolated along a 3D line at a fractional location.
- **ST_3DLongestLine** - 3D 几何体中最长的线。
- **ST_3DMaxDistance** - 3D 几何体之间的最大距离 (SRS 无关) 3D 几何体之间的最大距离。
- **ST_3DPerimeter** - 3D 几何体的周长。
- **ST_3DShortestLine** - 3D 几何体中最短的线。
- **ST_3DUnion** - Perform 3D union.
- **ST_AddMeasure** - Interpolates measures along a linear geometry.
- **ST_AddPoint** - 3D 几何体添加点。
- **ST_Affine** - Apply a 3D affine transformation to a geometry.
- **ST_ApproximateMedialAxis** - 3D 几何体的近似中轴。
- **ST_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST_AsEWKB** - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- **ST_AsEWKT** - 返回 WKT(Well-Known Text) 格式 SRID 几何体。
- **ST_AsGML** - 返回 GML 2 或 GML 3 格式几何体。
- **ST_AsGeoJSON** - Return a geometry or feature in GeoJSON format.

- **ST_AsHEXEWKB** - 将几何体 (NDR) 或几何体 (XDR) 转换为 HEXEWKB (二进制) 格式。
- **ST_AsKML** - 将几何体 GML 2 或 GML 3 转换为 KML 格式。
- **ST_AsX3D** - 将几何体 X3D XML 格式: 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_CatmullRomSmoothing** - Returns a smoothed version of a geometry, using the Catmull-Rom spline algorithm
- **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** - 将几何体转换为点集合。
- **ST_DumpRings** - Returns a set of geometry_dump rows for the exterior and interior rings of a Polygon.
- **ST_DumpSegments** - 将几何体转换为线段集合。
- **ST_EndPoint** - Returns the last point of a LineString, CircularLineString, or NURBS Curve.
- **ST_ExteriorRing** - 返回多边形的外环。
- **ST_Extrude** - 将几何体沿 Z 轴拉伸。
- **ST_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
- **ST_Force2D** - 将几何体强制为 2D。
- **ST_ForceCurve** - 将几何体强制为曲线, 并指定曲线类型 (upcast)。
- **ST_ForceLHR** - LHR(Left Hand Reverse; 左手反向) 强制几何体。
- **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** - 强制几何体 (orientation) 符合右手定则 (Right-Hand Rule) 的几何体。
- **ST_ForceSFS** - 强制 SFS 1.1 的几何体。
- **ST_Force3D** - 强制 XYZ 的几何体。ST_Force3DZ 强制 3DZ 的几何体。
- **ST_Force3DZ** - 强制 XYZ 的几何体。
- **ST_Force4D** - 强制 XYZM 的几何体。
- **ST_ForceCollection** - 强制集合几何体。
- **ST_GeomFromEWKB** - EWKB(Extended Well-Known Binary) 格式的 ST_Geometry 几何体。
- **ST_GeomFromEWKT** - EWKT(Extended Well-Known Text) 格式的 ST_Geometry 几何体。
- **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** - 几何体的中位数 (median) 几何体。
- **ST_GeometryN** - ST_Geometry 几何体的第 N 个几何体。
- **ST_GeometryType** - ST_Geometry 几何体的类型。
- **ST_HasArc** - Tests if a geometry contains a circular arc
- **ST_HasM** - Checks if a geometry has an M (measure) dimension.
- **ST_HasZ** - Checks if a geometry has a Z dimension.
- **ST_InteriorRingN** - 几何体的第 N 个内环。
- **ST_InterpolatePoint** - 沿几何体 (M 值) 插值点。
- **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。多线环 (多边形) 是否是 TRUE 或 FALSE。
- **ST_IsCollection** - 是否是集合几何体, 点, 线, 面 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** - 是否是简单几何体 TRUE 或 FALSE。
- **ST_IsSolid** - 是否是实心几何体。是否是实心几何体。
- **ST_IsValidTrajectory** - Tests if the geometry is a valid trajectory.
- **ST_LengthSpheroid** - 球面几何体的长度。
- **ST_LineFromMultiPoint** - 从多点生成线。
- **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_MMax** - Returns the M maxima of a geometry.
- **ST_MMin** - Returns the M minima of a geometry.
- **ST_MSize** - Returns the M size of a geometry.
- **ST_MakeLine** - 从点数组中创建 LineString 几何。
- **ST_MakePoint** - Creates a 2D, 3DZ or 4D Point.
- **ST_MakePolygon** - Creates a Polygon or CurvePolygon from a shell and optional list of holes.
- **ST_MakeSolid** - 从点数组中创建 Polygon 几何。如果指定了 true，则返回 Polygon 的 Solid 表示形式。如果指定了 false，则返回 Polygon 的 Shell 表示形式。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** - 几何中的点数量 (包括空值)。
- **ST_NRings** - 几何中的环数量。
- **ST_Node** - Nodes a collection of lines.
- **ST_NumCurves** - Return the number of component curves in a CompoundCurve.
- **ST_NumGeometries** - 几何集合中的几何数量。
- **ST_NumPatches** - 几何集合中的补丁数量。如果为 NULL，则返回 0。
- **ST_Orientation** - 几何的 (orientation) 值。
- **ST_PatchN** - ST_Geometry 的补丁 N。
- **ST_PointFromWKB** - 从 SRID 和 WKB 数据创建点。
- **ST_PointN** - ST_LineString 或 ST_CircularString 中的点 N。
- **ST_PointOnSurface** - Computes a point guaranteed to lie in a polygon, or on a geometry.
- **ST_Points** - 从几何中提取点集合。
- **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** - 反转几何。
- **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** - 将几何体中的指定点替换为新的点。
- **ST_ShiftLongitude** - Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- **ST_SnapToGrid** - 将几何体中的点 snap 到指定的栅格。
- **ST_StartPoint** - Returns the first point of a LineString, CircularLineString, or NURBSCurve.
- **ST_StraightSkeleton** - 计算几何体的 (straight skeleton) 骨架。
- **ST_SwapOrdinates** - 交换几何体中的经纬度坐标。
- **ST_SymDifference** - Computes a geometry representing the portions of geometries A and B that do not intersect.
- **ST_Tessellate** - 将几何体 tessellate (tessellation) 成 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** - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **ST_WrapX** - X 坐标值在 -180 到 180 度之间。
- **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_XSize** - Returns the X size 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_YSize** - Returns the Y size of a 2D or 3D bounding box or a geometry.
- **ST_Z** - Returns the Z coordinate of a Point.
- **ST_ZMFlag** - ST_Geometry 是否包含 M 坐标。
- **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_ZSize** - Returns the Z size of a 2D or 3D bounding box or a geometry.
- **Equals** - 判断两个 TopoGeometry 是否相等。
- **Intersects** - 判断两个 TopoGeometry 是否相交。

- **UpdateGeometrySRID** - Updates the SRID of all features in a geometry column, and the table meta-data.
- **&&&** - A n 维 B n 维 TRUE.
- **&&&(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 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) SRID 几何。
- **ST_AsHEXEWKB** - 返回 (NDR) (XDR) HEXEWKB (几何) 数据。
- **ST_AsSVG** - Returns SVG path data for a geometry.
- **ST_AsText** - 返回 WKT(Well-Known Text) SRID 几何。
- **ST_ClusterDBSCAN** - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- **ST_ClusterWithin** - Aggregate function that clusters geometries by separation distance.
- **ST_ClusterWithinWin** - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- **ST_Collect** - Creates a GeometryCollection or Multi* geometry from a set of geometries.

- **ST_CoordDim** - ST_Geometry 的维度。
- **ST_CurveToLine** - Converts a geometry containing curves to a linear geometry.
- **ST_Distance** - 返回 3 个 (最长) 距离。
- **ST_Dump** - Returns a set of geometry_dump rows for the components of a geometry.
- **ST_DumpPoints** - 返回几何体中的点。
- **ST_EndM** - Returns the M coordinate of the last point of a LineString, CircularLineString, or NURBS-Curve.
- **ST_EndPoint** - Returns the last point of a LineString, CircularLineString, or NURBS-Curve.
- **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** - 强制为 2D 几何体。
- **ST_ForceCurve** - 强制为曲线, 强制为曲线 (upcast) 几何体。
- **ST_ForceSFS** - 强制为 SFS 1.1 几何体。
- **ST_Force3D** - 强制为 XYZ 几何体。ST_Force3DZ 强制为 XYZ。
- **ST_Force3DM** - 强制为 XYM 几何体。
- **ST_Force3DZ** - 强制为 XYZ 几何体。
- **ST_Force4D** - 强制为 XYZM 几何体。
- **ST_ForceCollection** - 强制为集合几何体。
- **ST_GeoHash** - 返回 GeoHash 字符串。
- **ST_GeogFromWKB** - WKB 到 EWKB (WKB) 的转换。
- **ST_GeomFromEWKB** - EWKB (Extended Well-Known Binary) 到 ST_Geometry 的转换。
- **ST_GeomFromEWKT** - EWKT (Extended Well-Known Text) 到 ST_Geometry 的转换。
- **ST_GeomFromText** - WKT 到 ST_Geometry 的转换。
- **ST_GeomFromWKB** - WKB (Well-Known Binary) 到 SRID 几何体的转换。
- **ST_GeometryN** - ST_Geometry 的第 N 个几何体。
- **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **&<|** - A 与 B 的交集是否为 TRUE。
- **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 是否为闭合的。如果是 (是) 则为 TRUE。
- **ST_IsCollection** - 是否为集合, 是否为, 是否为 TRUE。
- **ST_IsEmpty** - Tests if a geometry is empty.
- **ST_LineToCurve** - Converts a linear geometry to a curved geometry.

- **ST_MMax** - Returns the M maxima of a geometry.
- **ST_MMin** - Returns the M minima of a geometry.
- **ST_MSize** - Returns the M size of a geometry.
- **ST_MemSize** - ST_Geometry 函数返回内存大小。
- **ST_NPoints** - 返回几何体中的点数量 (N)。
- **ST_NRings** - 返回几何体中的环数量。
- **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** - Returns the spatial reference identifier for a geometry.
- **ST_Scale** - Scales a geometry by given factors.
- **ST_SetEndM** - Sets the M coordinate of the last point of a LineString, CircularLineString, or NURBS-Curve.
- **ST_SetSRID** - Set the SRID on a geometry.
- **ST_SetStartM** - Sets the M coordinate of the first point of a LineString, CircularLineString, or NURBS-Curve.
- **ST_StartM** - Returns the M coordinate of the first point of a LineString, CircularLineString, or NURBS-Curve.
- **ST_StartPoint** - Returns the first point of a LineString, CircularLineString, or NURBS-Curve.
- **ST_Summary** - 返回几何体的摘要信息。
- **ST_SwapOrdinates** - 交换几何体中的坐标轴。
- **ST_TransScale** - Translates and scales a geometry by given offsets and factors.
- **ST_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST_Translate** - Translates a geometry by given offsets.
- **ST_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- **ST_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST_XSize** - Returns the X size 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_YSize** - Returns the Y size of a 2D or 3D bounding box or a geometry.
- **ST_ZMFlag** - ST_Geometry 函数返回 Z 标志。
- **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_ZSize** - Returns the Z size of a 2D or 3D bounding box or a geometry.
- **UpdateGeometrySRID** - Updates the SRID of all features in a geometry column, and the table meta-data.
- **~(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).
- **&&** - A 2D BOX2DF B 2D BOX2DF TRUE.
- **&&&** - A n BOX2DF B n BOX2DF 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 BOX2DF 0 BOX2DF.
- **CG_3DConvexHull** - BOX2DF.
- **CG_3DDifference** - 3 BOX2DF.
- **CG_3DIntersection** - 3 BOX2DF.

- **CG_3DUnion** - Perform 3D union using postgis_sfcgal.
- **CG_ApproximateMedialAxis** - 返回给定多边形或几何体的近似 medial axis.
- **CG_Extrude** - 将给定的 2D 几何体沿指定的高度和方向挤出.
- **CG_ForceLHR** - LHR(Left Hand Reverse; 左手规则) 强制多边形为左手规则.
- **CG_GenerateFlatRoof** - Generate a flat (box) roof from a footprint polygon.
- **CG_GenerateGableRoof** - Generate a gable roof from a footprint polygon.
- **CG_GenerateHippedRoof** - Generate a hipped roof from a footprint polygon.
- **CG_GenerateRoof** - Generate a roof of the requested type from a footprint polygon.
- **CG_GenerateSkillionRoof** - Generate a skillion (single-slope) roof from a footprint polygon.
- **CG_IsPlanar** - 检查给定的 3D 几何体是否是平面的.
- **CG_IsSolid** - 检查给定的 3D 几何体是否是实体的. 返回 true 表示是实体, false 表示不是.
- **CG_MakeSolid** - 将给定的 3D 几何体转换为实体. 如果输入是 TIN, 则返回 TIN; 否则返回多面体.
- **CG_PolygonRepair** - Repair an invalid polygon or multipolygon.
- **CG_StraightSkeleton** - 计算给定多边形的 (straight skeleton) 骨架.
- **CG_Tessellate** - 将给定的 3D 几何体进行三角剖分 (tessellation) 转换为 TIN 格式.
- **CG_Visibility** - Compute a visibility polygon from a point or a segment in a polygon geometry
- **CG_Volume** - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **GeometryType** - ST_Geometry 的几何类型.
- **ST_3DArea** - 3D 几何体的面积. 返回 0 表示不是封闭表面.
- **ST_3DClosestPoint** - g2 几何体到 g1 几何体的 3D 最短距离. 返回 3D 最短距离.
- **ST_3DConvexHull** - 计算 3D 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** - 3D 几何体的差集.
- **ST_3DDistance** - 计算 3D 几何体之间的距离 (SRS 无关) 3D 最短距离.
- **ST_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST_3DIntersection** - 3D 几何体的交集.
- **ST_3DIntersects** - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- **ST_3DLongestLine** - 计算 3D 几何体的 (longest) 最长线.
- **ST_3DMaxDistance** - 计算 3D 几何体之间的最大距离 (SRS 无关) 3D 最长距离.
- **ST_3DShortestLine** - 计算 3D 几何体的 (shortest) 最短线.

- **ST_3DUnion** - Perform 3D union.
- **ST_Affine** - Apply a 3D affine transformation to a geometry.
- **ST_ApproximateMedialAxis** - 返回几何的近似 medial axis.
- **ST_Area** - 返回几何的面积.
- **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.
- **ST_AsGML** - 返回 GML 2 或 GML 3 格式的几何.
- **ST_AsX3D** - 返回 X3D XML 格式的几何: ISO-IEC-19776-1.2-X3DEncodings-XML 格式.
- **ST_CoordDim** - ST_Geometry 的坐标维度.
- **ST_Dimension** - ST_Geometry 的维度.
- **ST_Dump** - Returns a set of geometry_dump rows for the components of a geometry.
- **ST_DumpPoints** - 返回几何的点的集合.
- **ST_Expand** - Returns a bounding box expanded from another bounding box or a geometry.
- **ST_Extent** - Aggregate function that returns the bounding box of geometries.
- **ST_Extrude** - 返回几何的 extruded 版本.
- **ST_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
- **ST_Force2D** - 强制为 2D 几何.
- **ST_ForceLHR** - LHR(Left Hand Reverse; 左手规则) 几何.
- **ST_ForceRHR** - 强制为 RHR (orientation) 几何 (Right-Hand Rule 右手规则) 几何.
- **ST_ForceSFS** - 强制为 SFS 1.1 格式的几何.
- **ST_Force3D** - 强制为 XYZ 格式的几何. ST_Force3DZ 强制为 3DZ 格式.
- **ST_Force3DZ** - 强制为 XYZ 格式的几何.
- **ST_ForceCollection** - 强制为集合类型的几何.
- **ST_GeomFromEWKB** - EWKB(Extended Well-Known Binary) 格式的 ST_Geometry 几何.
- **ST_GeomFromEWKT** - EWKT(Extended Well-Known Text) 格式的 ST_Geometry 几何.
- **ST_GeomFromGML** - 从 GML 格式返回 PostGIS 几何.
- **ST_GeometryN** - ST_Geometry 的第 N 个几何.
- **ST_GeometryType** - ST_Geometry 的几何类型.
- **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **&<|** - A 几何 B 几何 返回 TRUE 如果 A 包含 B.
- **~=** - A 几何 B 几何 返回 TRUE 如果 A 与 B 相交.

- **ST_IsClosed** - LINESTRING 是否是闭合的。如果是，返回 TRUE。如果是非闭合的 (LINESTRING 不是闭合的) 返回 FALSE。
- **ST_IsPlanar** - 是否是平面几何。
- **ST_IsSolid** - 是否是实心几何。如果是，返回 TRUE。如果是非实心的，返回 FALSE。
- **ST_MakeSolid** - 将非实心几何转换为实心几何。如果是非实心的，返回非实心几何；如果是实心的，返回实心几何。
- **ST_MemSize** - ST_Geometry 的内存大小。
- **ST_NPoints** - 几何中的点数量 (POINT) 返回点数量。
- **ST_NumGeometries** - 几何中的几何数量。如果是多几何体，返回数量。
- **ST_NumPatches** - 几何中的补丁数量。如果是 NULL，返回 NULL。
- **ST_PatchN** - ST_Geometry 的补丁 N。
- **ST_RemoveRepeatedPoints** - Returns a version of a geometry with duplicate points removed.
- **ST_Reverse** - 反转几何。
- **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** - 几何的 (straight skeleton) 骨架。
- **ST_Summary** - 几何的摘要。
- **ST_SwapOrdinates** - 交换几何的纵坐标。
- **ST_Tessellate** - 几何的 (tessellation) 三角化。如果是 TIN，返回 TIN；如果不是，返回非 TIN。
- **ST_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST_Volume** - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **~(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (BOX2DF).
- **&&** - A 2D 几何 B 2D 几何 是否是 TRUE。
- **&&&** - A n 几何 B n 几何 是否是 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.

13.11 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 srld" 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_LineFromMPoint							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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	✓						
ST_Letters	✓						
ST_MakeNurbsCurve	✓						
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_InteriorRing	✓		✓		✓		
ST_NumCurves	✓		✓		✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_CurveN	✓		✓		✓		
ST_IsClosed	✓		✓	✓	✓	✓	
ST_IsCollection	✓		✓	✓			
ST_IsEmpty	✓			✓	✓		
ST_IsPolygonCC	✓		✓				
ST_IsPolygonCV	✓		✓				
ST_IsRing	✓				✓		
ST_IsSimple	✓		✓		✓		
ST_M	✓		✓		✓		
ST_MemSize	✓		✓	✓		✓	✓
ST_NDims	✓		✓				
ST_NPoints	✓		✓	✓		✓	
ST_NRings	✓		✓	✓			
ST_NumGeometries	✓		✓		✓	✓	✓
ST_NumInteriorRings	✓				✓		
ST_NumInteriorRing	✓						
ST_NumPatches	✓		✓		✓	✓	
ST_NumPoints	✓				✓		
ST_PatchN	✓		✓		✓	✓	
ST_PointN	✓		✓	✓	✓		
ST_Points	✓		✓	✓			
ST_StartPoint	✓		✓	✓	✓		
ST_StartM	✓			✓			
ST_EndM	✓			✓			
ST_SetStartM	✓			✓			
ST_SetEndM	✓			✓			
ST_Summary	✓	✓		✓		✓	✓
ST_X	✓		✓		✓		
ST_Y	✓		✓		✓		
ST_Z	✓		✓		✓		
ST_ZMFlag	✓		✓	✓			
ST_HasZ	✓		✓				
ST_HasM	✓		✓				
ST_ControlPoint	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Degree	✓						
ST_Weights	✓						
ST_Knots	✓						
ST_NumControlPoints	✓						
ST_NurbsCurve	✓	ational					
ST_IsNurbsCurve	✓						
ST_Evaluate	✓						
ST_NurbsToLine	✓	ring					
ST_AddPoint	✓		✓				
ST_CollectionExtract	✓	ct					
ST_CollectionHomogenize	✓						
ST_CurveToLine	✓		✓	✓	✓		
ST_Scroll	✓		✓				
ST_FlipCoordinates	✓		✓	✓		✓	✓
ST_Force2D	✓		✓	✓		✓	
ST_Force3D	✓		✓	✓		✓	
ST_Force3DZ	✓		✓	✓		✓	
ST_Force3DM	✓			✓			
ST_Force4D	✓		✓	✓			
ST_ForceCollection	✓		✓	✓		✓	
ST_ForceCurve	✓		✓	✓			
ST_ForcePolygonCW	✓		✓				
ST_ForcePolygonCCW	✓		✓				
ST_ForceSFS	✓		✓	✓		✓	✓
ST_ForceRHR	✓		✓			✓	
ST_LineExtend	✓						
ST_LineToCurve	✓		✓	✓			
ST_Multi	✓						
ST_Normalize	✓						
ST_Project	✓	✓					
ST_QuantizeCoordinates	✓						
ST_RemovePoint	✓		✓				
ST_RemoveRepeatedPoints	✓		✓			✓	
ST_RemoveIrrelevantPointsForView	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_RemoveSmallParts	✓						
ST_Reverse	✓		✓			✓	
ST_Segmentize	✓	✓					
ST_SetPoint	✓		✓				
ST_ShiftLongitude	✓		✓			✓	✓
ST_WrapX	✓		✓				
ST_SnapToGrid	✓		✓				
ST_Snap	✓						
ST_SwapOrdinate	✓		✓	✓		✓	✓
ST_IsValid	✓				✓		
ST_IsValidDetail	✓						
ST_IsValidReason	✓						
ST_MakeValid	✓		✓				
ST_InverseTransformPipeline	✓						
ST_SetSRID	✓			✓			
ST_SRID	✓			✓	✓		
ST_Transform	✓			✓	✓	✓	
ST_TransformPipeline	✓						
postgis_srs_codes							
postgis_srs							
postgis_srs_all							
postgis_srs_search	✓						
ST_BdPolyFromText	✓						
ST_BdMPolyFromText	✓						
ST_GeogFromText		✓					
ST_GeographyFromText		✓					
ST_GeomCollFromText	✓				✓		
ST_GeomFromEWKT			✓	✓		✓	✓
ST_GeomFromMRC21	✓						
ST_GeometryFromText	✓				✓		
ST_GeomFromI	✓			✓	✓		
ST_LineFromText	✓				✓		
ST_MLineFromText	✓				✓		
ST_MPointFromText	✓				✓		
ST_MPolyFromText	✓				✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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_GeomFromTWKB	✓						
ST_GMLToSQL	✓				✓		
ST_LineFromEncodedPolyline	✓						
ST_PointFromGeoHash							
ST_FromFlatGeobufToTable							
ST_FromFlatGeobuf							
ST_AsEWKT	✓	✓	✓	✓		✓	✓
ST_AsText	✓	✓		✓	✓		
ST_AsBinary	✓	✓	✓	✓	✓	✓	✓
ST_AsEWKB	✓		✓	✓		✓	✓
ST_AsHEXEWKB	✓		✓	✓			
ST_AsEncodedPolyline	✓						
ST_AsFlatGeobuf	✗						
ST_AsGeobuf	✗						
ST_AsGeoJSON	✓	✓	✓				
ST_AsGML	✓	✓	✓		✓	✓	✓
ST_AsKML	✓	✓	✓				
ST_AsLatLonText	✓						
ST_AsMARC21	✓						
ST_AsMVTGeom	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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)	✓			✓		✓	
~(box2df,box2df)	￼			✓		✓	
~=	✓					✓	
<->	✓	✓					
=	✓						
<#>	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
<<->>	✓						
ST_3DIntersect	✓		✓		✓	✓	✓
ST_Contains	✓				✓		
ST_ContainsProperly	✓						
ST_CoveredBy	✓	✓					
ST_Covers	✓	✓					
ST_Crosses	✓				✓		
ST_Disjoint	✓				✓		
ST_Equals	✓				✓		
ST_Intersects	✓	✓		✓	✓		✓
ST_LineCrossingDirection	✓						
ST_OrderingEquivalent	✓				✓		
ST_Overlaps	✓				✓		
ST_Relate	✓				✓		
ST_RelateMatch							
ST_Touches	✓				✓		
ST_Within	✓				✓		
ST_3DDWithin	✓		✓		✓	✓	
ST_3DDFullyWithin	✓		✓			✓	
ST_DFullyWithin	✓						
ST_DWithin	✓	✓					
ST_PointInsideCurve	✓						
ST_Area	✓	✓			✓	✓	
ST_Azimuth	✓	✓					
ST_Angle	✓						
ST_ClosestPoint	✓	✓					
ST_3DClosestPoint	✓		✓			✓	
ST_Distance	✓	✓		✓	✓		
ST_3DDistance	✓		✓		✓	✓	
ST_DistanceSphere	✓						
ST_DistanceSphereoid	✓						
ST_FrechetDistance	✓						
ST_HausdorffDistance	✓						
ST_Length	✓	✓			✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Length2D	✓						
ST_3DLength	✓		✓		✓		
ST_LengthSphere	✓		✓				
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_CatmullRomSmoothing	✓		✓				
ST_ConcaveHull	✓						
ST_ConvexHull	✓		✓		✓		
ST_DelaunayTriangles	✓		✓				✓
ST_FilterByM	✓						
ST_GeneratePoints	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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_CoverageClean	✓						
ST_CoverageSimplify	✓						
ST_CoverageEdges	✓						
ST_CoverageUnion	✓						
ST_Affine	✓		✓	✓		✓	✓
ST_Rotate	✓		✓	✓		✓	✓
ST_RotateX	✓		✓			✓	✓
ST_RotateY	✓		✓			✓	✓
ST_RotateZ	✓		✓	✓		✓	✓
ST_Scale	✓		✓	✓		✓	✓
ST_Translate	✓		✓	✓			
ST_TransScale	✓		✓	✓			
ST_ClusterDBSCAN	✓			✓			

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_ClusterInter	✓ting						
ST_ClusterInter	✓tingWin						
ST_ClusterRelat	✓in						
ST_MinimumSp	✓ingTree						
ST_ClusterKMe	✓						
ST_ClusterWith	✓			✓			
ST_ClusterWith	✓/in			✓			
Box2D	✓			✓		✓	✓
Box3D	✓		✓	✓		✓	✓
ST_EstimatedEx	✓t			✓			
ST_Expand	✓					✓	✓
ST_Extent	✓					✓	✓
ST_3DExtent	✓		✓	✓		✓	✓
ST_MakeBox2D	✓						
ST_3DMakeBox	✓						
ST_XMax	✓		✓	✓			
ST_XMin	✓		✓	✓			
ST_YMax	✓		✓	✓			
ST_YMin	✓		✓	✓			
ST_ZMax	✓		✓	✓			
ST_ZMin	✓		✓	✓			
ST_XSize	✓		✓	✓			
ST_YSize	✓		✓	✓			
ST_ZSize	✓		✓	✓			
ST_MMin	✓		✓	✓			
ST_MMax	✓		✓	✓			
ST_MSize	✓		✓	✓			
ST_LineInterpol	✓Point	✓	✓				
ST_3DLineInter	✓atePoint		✓				
ST_LineInterpol	✓Points	✓	✓				
ST_LineLocateP	✓t	✓					
ST_LineSubstri	✓	✓	✓				
ST_LocateAlong	✓				✓		
ST_LocateBetw	✓				✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_LocateBetweenElevations	✓		✓				
ST_InterpolatePoint	✓		✓				
ST_3DInterpolatePoint	✓		✓				
ST_AddMeasure	✓		✓				
ST_IsValidTrajectory	✓		✓				
ST_ClosestPointApproach	✓		✓				
ST_DistanceCPA	✓		✓				
ST_CPAWithin	✓		✓				
postgis.gdal_datapath							
postgis.gdal_enabled_drivers							
postgis.enable_outdb_rasters							
postgis.gdal_vsi_options							
postgis.gdal_cpl_debug							
PostGIS_AddBB	✓			✓			
PostGIS_DropBB	✓			✓			
PostGIS_HasBB	✓			✓			
postgis_sfcgal_version							
postgis_sfcgal_full_version							
CG_ForceLHR	￼		￼			￼	￼
CG_IsPlanar	￼		￼			￼	￼
CG_IsSolid	￼		￼			￼	￼
CG_MakeSolid	￼		￼			￼	￼
CG_Orientation	￼		￼				
CG_Area	￼						
CG_3DArea	￼		￼		￼	￼	￼
CG_Volume	￼		￼		￼	￼	￼
ST_ForceLHR	￼		￼			￼	￼
ST_IsPlanar	￼		￼			￼	￼
ST_IsSolid	￼		￼			￼	￼
ST_MakeSolid	￼		￼			￼	￼
ST_Orientation	￼		￼				

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_3DArea							
ST_Volume							
CG_Intersectio							
CG_Intersects							
CG_3DIntersec							
CG_Difference							
ST_3DDifferen							
CG_3DDifferen							
CG_Distance							
CG_3DDistance							
ST_3DConvexH							
CG_3DConvexH							
ST_3DIntersec							
CG_3DIntersec							
CG_Union							
ST_3DUnion							
CG_3DUnion							
ST_AlphaShape	✔						
CG_AlphaShap							
CG_ApproxCon	partition						
ST_Approximat	dialAxis						
CG_Approxima	dialAxis						
CG_GenerateFl	of						
CG_GenerateH	dRoof						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
CG_GenerateG	Roof						
CG_GenerateSI	nRoof						
CG_GenerateR							
CG_PolygonRej							
ST_Constrained	aunayTriangles						
CG_Constrained	aunayTriangles						
ST_Extrude							
CG_Extrude							
CG_ExtrudeStr	tSkeleton						
CG_GreeneApp	onvexPartition						
ST_Minkowski							
CG_Minkowski							
ST_OptimalAlp	ape						
CG_OptimalAlp	ape						
CG_OptimalCo	Partition						
CG_StraightSk	n						
ST_StraightSk	n						
ST_Tessellate							
CG_Tessellate							
CG_Triangulate							
CG_Visibility							
CG_YMonotone	ition						
CG_StraightSk	nPartition						
CG_3DBuffer							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
CG_Rotate							
CG_2DRotate							
CG_3DRotate							
CG_RotateX							
CG_RotateY							
CG_RotateZ							
CG_Scale							
CG_3DScale							
CG_3DScaleAr		Center					
CG_Translate							
CG_3DTranslat							
CG_Simplify							
CG_3DAlphaW		ng					
CG_NurbsCurv		erpolate					
CG_NurbsCurv		roximate					
CG_NurbsCurv		ivative					
getfaceedges_returntype							
TopoGeometry							
validatetopology_returntype							
TopoElement							
TopoElementArray							
AddTopoGeometryColumn							
RenameTopoGeometryColumn							
DropTopology							
RenameTopology							
DropTopoGeometryColumn							
FixCorruptTopoGeometryColumn							
Populate_Topology_Layer							
TopologySummary							
ValidateTopolog	✓						
ValidateTopologyRelation							
ValidateTopolog	✓	recision					
MakeTopologyP	✓	ise					

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
FindTopology	✓						
FindLayer	✓						
TotalTopologySize							
UpgradeTopology							
CreateTopology							
CopyTopology							
ST_InitTopoGeo					✓		
ST_CreateTopoGeo	✓				✓		
TopoGeo_AddPoint	✓						
TopoGeo_AddLineString	✓						
TopoGeo_AddPolygon	✓						
TopoGeo_LoadGeometry	✓						
ST_AddIsoNode	✓				✓		
ST_AddIsoEdge	✓				✓		
ST_AddEdgeNewFaces	✓				✓		
ST_AddEdgeModifyFace	✓				✓		
ST_RemEdgeNewFace					✓		
ST_RemEdgeModFace					✓		
ST_ChangeEdgeGeometry	✓				✓		
ST_ModEdgeSplit	✓				✓		
ST_ModEdgeHeal					✓		
ST_NewEdgeHeal					✓		
ST_MoveIsoNode	✓				✓		
ST_NewEdgesSplit	✓				✓		
ST_RemoveIsoNode					✓		
ST_RemoveIsoEdge					✓		
GetEdgeByPoint	✓						
GetFaceByPoint	✓						
GetFaceContainsPoint	✓						
GetNodeByPoint	✓						
GetTopologyID							
GetTopologySRID							
GetTopologyName							
ST_GetFaceEdges					✓		
ST_GetFaceGeometry	✓				✓		
GetRingEdges							
GetNodeEdges							
FindVertexSegmentPairsBelowDistance							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
Polygonize							
AddNode	✓						
AddEdge	✓						
AddFace	✓						
ST_Simplify	✓						
RemoveUnused	✓ natives						
CreateTopoGeom	✓						
toTopoGeom	✓						
TopoElementArray_Agg							
TopoElement	✓						
clearTopoGeom	✓						
TopoGeom_add	✓ nent						
TopoGeom_remove	✓ ment						
TopoGeom_add	✓ oGeom						
toTopoGeom							
GetTopoGeomElementArray							
GetTopoGeomElements							
ST_SRID	✓				✓		
AsGML	✓						
AsTopoJSON	✓						
Equals	✓		✓				
Intersects	✓		✓				
geomval							
addbandarg							
rastbandarg							
raster							
reclassarg							
summarystats							
unionarg							
AddRasterConstraints							
DropRasterConstraints							
AddOverviewConstraints							
DropOverviewConstraints							
PostGIS_GDAL_Version							
PostGIS_Raster_Lib_Build_Date							
PostGIS_Raster_Lib_Version							
ST_GDALDrivers							
UpdateRasterSRID							
ST_CreateOverview							
ST_AddBand							
ST_AsRaster	✓						
ST_AsRasterAgg	✓						
ST_Band							
ST_MakeEmptyCoverage							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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_WorldToRasterCoord	✓						
ST_WorldToRasterCoordX	✓						
ST_WorldToRasterCoordY	✓						
ST_BandMetaData							
ST_BandNoDataValue							
ST_BandIsNoData							
ST_BandPath							
ST_BandFileSize							
ST_BandFileTimestamp							
ST_BandPixelType							
ST_MinPossibleValue							
ST_HasNoBand							
ST_PixelAsPolygon	✓						
ST_PixelAsPolygons							
ST_PixelAsPoint	✓						
ST_PixelAsPoints							
ST_PixelAsCentroid	✓ 1						
ST_PixelAsCentroids							
ST_Value	✓						
ST_NearestValue	✓						
ST_SetZ	✓						
ST_SetM	✓						
ST_Neighborhood	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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							
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_ReclassExact							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Union							
ST_Distinct4ma							
ST_InvDistWeight4ma							
ST_Max4ma							
ST_Mean4ma							
ST_Min4ma							
ST_MinDist4ma							
ST_Range4ma							
ST_StdDev4ma							
ST_Sum4ma							
ST_Aspect							
ST_HillShade							
ST_Roughness							
ST_Slope							
ST_TPI							
ST_TRI							
ST_InterpolateF✓er							
ST_Contour							
Box3D							
ST_ConvexHull✓							
ST_DumpAsPolygons							
ST_Envelope✓							
ST_MinConvexH✓							
ST_Polygon✓							
ST_Intersection✓ctions							
&&✓							
&<							
&>							
=							
@✓							
~=							
~✓							
ST_Contains							
ST_ContainsProperly							
ST_Covers							
ST_CoveredBy							
ST_Disjoint							
ST_Intersects✓							
ST_Overlaps							
ST_Touches							
ST_SameAlignment							
ST_NotSameAlignmentReason							
ST_Within							
ST_DWithin							
ST_DFullyWithin							

13.12 New, Enhanced or changed PostGIS Functions

13.12.1 PostGIS Functions new or enhanced in 3.7

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.7

- **CG_ApproximateMedialAxis** - Availability: 3.7.0 - projected parameter. Requires SFCGAL >= 2.3.0 for projected result; falls back to non-projected with a notice on older versions.
- **CG_GenerateFlatRoof** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a flat (box) roof from a footprint polygon.
- **CG_GenerateGableRoof** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a gable roof from a footprint polygon.
- **CG_GenerateHippedRoof** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a hipped roof from a footprint polygon.
- **CG_GenerateRoof** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a roof of the requested type from a footprint polygon.
- **CG_GenerateSkillionRoof** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a skillion (single-slope) roof from a footprint polygon.
- **CG_NurbsCurveApproximate** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Creates an approximating NURBS curve fitting data points within a tolerance
- **CG_NurbsCurveDerivative** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter
- **CG_NurbsCurveInterpolate** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Creates an interpolating NURBS curve passing through all given data points
- **CG_PolygonRepair** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0 and CGAL >= 6.0. Repair an invalid polygon or multipolygon.
- **FindVertexSegmentPairsBelowDistance** - Availability: 3.7 Find pairs of topology vertex/segment that are closer than tolerated distance
- **ST_3DInterpolatePoint** - Availability: 3.7.0 Returns the interpolated measure of a geometry closest to a point in 3D.
- **ST_ClusterRelateWin** - Availability: 3.7.0 Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.
- **ST_ControlPoints** - Availability: 3.7.0 Returns the control points of a NURBS curve as a MULTIPOINT geometry.
- **ST_CoverageEdges** - Availability: 3.7.0 Computes the unique edges of a polygonal coverage.
- **ST_Degree** - Availability: 3.7.0 Returns the polynomial degree of a NURBS curve.
- **ST_Evaluate** - Availability: 3.7.0 Evaluates a NURBS curve at a specific parameter value and returns the resulting point.
- **ST_IsNurbsCurve** - Availability: 3.7.0 Checks if a geometry is a NURBS curve.
- **ST_Knots** - Availability: 3.7.0 Returns the knot vector of a NURBS curve.

- **ST_AsGeoJSON** - Changed: 3.7.0 added warning about duplicate keys Return a geometry or feature in GeoJSON format.
- **toTopoGeom** - Changed: 3.7.0 changes meaning of tolerance 0 to really mean 0 and -1 (new default) to request using the topology precision. Converts a simple Geometry into a topo geometry.

13.12.2 PostGIS Functions new or enhanced in 3.6

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.6

- **CG_2DRotate** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry by a given angle around a specified point in 2D.
 - **CG_3DAlphaWrapping** - Availability: 3.6.0 - requires SFCGAL >= 2.1.0 Computes a 3D Alpha-wrapping strictly enclosing a geometry.
 - **CG_3DBuffer** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Computes a 3D buffer around a geometry.
 - **CG_3DRotate** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry in 3D space around an axis vector.
 - **CG_3DScale** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Scales a geometry by separate factors along X, Y, and Z axes.
 - **CG_3DScaleAroundCenter** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Scales a geometry in 3D space around a specified center point.
 - **CG_3DTranslate** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Translates (moves) a geometry by given offsets in 3D space.
 - **CG_Rotate** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry by a given angle around the origin (0,0).
 - **CG_RotateX** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry around the X-axis by a given angle.
 - **CG_RotateY** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry around the Y-axis by a given angle.
 - **CG_RotateZ** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry around the Z-axis by a given angle.
 - **CG_Scale** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Scales a geometry uniformly in all dimensions by a given factor.
 - **CG_Simplify** - Availability: 3.6.0 - requires SFCGAL >= 2.1.0 Reduces the complexity of a geometry while preserving essential features and Z/M values.
 - **CG_StraightSkeletonPartition** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0. Computes the straight skeleton partition of a polygon.
 - **CG_Translate** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Translates (moves) a geometry by given offsets in 2D space.
 - **FixCorruptTopoGeometryColumn** - Availability: 3.6.1 Fixes topogeometry corruption caused by upgrade to postgis_topology 3.6.0 and higher
 - **MakeTopologyPrecise** - Availability: 3.6.0 Snap topology vertices to precision grid.
 - **ST_AsRasterAgg** - Availability: 3.6.0 Aggregate. Renders PostGIS geometries into a new raster.
 - **ST_CatmullRomSmoothing** - Availability: 3.6.0 Returns a smoothed version of a geometry, using the Catmull-Rom spline algorithm
 - **ST_CoverageClean** - Availability: 3.6.0 - requires GEOS >= 3.14.0 Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.
-

- **ST_IntersectionFractions** - Availability: 3.6.0 Requires GEOS 3.14 or higher. Calculates the fraction of each raster cell that is covered by a given geometry.
- **ST_ReclassExact** - Availability: 3.6.0 Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.
- **TotalTopologySize** - Availability: 3.6.0 Total disk space used by the specified topology, including all indexes and TOAST data.
- **UpgradeTopology** - Availability: 3.6.0 Upgrades the specified topology to support large ids (int8) for topology and primitive ids.
- **ValidateTopologyPrecision** - Availability: 3.6.0 Returns non-precise vertices in the topology.
- **postgis.gdal_cpl_debug** - Availability: 3.6.0 A boolean configuration to turn logging of GDAL debug messages on and off.

Functions enhanced in PostGIS 3.6

- **CreateTopology** - Enhanced: 3.6.0 added topoid and usesLargeIDs parameters. Creates a new topology schema and registers it in the topology.topology table.

13.12.3 PostGIS Functions new or enhanced in 3.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.5

- **CG_3DArea** - Availability: 3.5.0 3 ￼￼￼￼￼￼￼￼￼￼. ￼￼￼ 0 ￼￼￼￼￼￼.
- **CG_3DConvexHull** - Availability: 3.5.0 ￼￼￼￼￼￼￼￼￼￼.
- **CG_3DDifference** - Availability: 3.5.0 3 ￼￼￼￼￼￼.
- **CG_3DDistance** - Availability: 3.5.0 Computes the minimum 3D distance between two geometries
- **CG_3DIntersection** - Availability: 3.5.0 3 ￼￼￼￼￼￼.
- **CG_3DIntersects** - Availability: 3.5.0 Tests if two 3D geometries intersect
- **CG_3DUnion** - Availability: 3.5.0 Perform 3D union using postgis_sfcgal.
- **CG_AlphaShape** - Availability: 3.5.0 - requires SFCGAL >= 1.4.1. Computes an Alpha-shape enclosing a geometry
- **CG_ApproxConvexPartition** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Computes approximal convex partition of the polygon geometry
- **CG_ApproximateMedialAxis** - Availability: 3.5.0 ￼￼￼￼￼￼￼￼￼￼.
- **CG_Area** - Availability: 3.5.0 Calculates the area of a geometry
- **CG_Difference** - Availability: 3.5.0 Computes the geometric difference between two geometries
- **CG_Distance** - Availability: 3.5.0 Computes the minimum distance between two geometries
- **CG_Extrude** - Availability: 3.5.0 ￼￼￼￼￼￼￼￼￼.
- **CG_ExtrudeStraightSkeleton** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Straight Skeleton Extrusion
- **CG_ForceLHR** - Availability: 3.5.0 LHR(Left Hand Reverse; ￼￼￼) ￼￼￼￼￼.

- **CG_GreeneApproxConvexPartition** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Computes approximal convex partition of the polygon geometry
- **CG_Intersection** - Availability: 3.5.0 Computes the intersection of two geometries
- **CG_Intersects** - Availability: 3.5.0 Tests if two geometries intersect (they have at least one point in common)
- **CG_IsPlanar** - Availability: 3.5.0 检查几何体是否是平面。
- **CG_IsSolid** - Availability: 3.5.0 检查几何体是否是实心体。检查几何体是否是实心体。
- **CG_MakeSolid** - Availability: 3.5.0 检查几何体是否是实心体。检查几何体是否是实心体。检查几何体是否是实心体，检查几何体是否是实心体 TIN 检查几何体是否是实心体。
- **CG_MinkowskiSum** - Availability: 3.5.0 检查几何体是否是实心体。
- **CG_OptimalAlphaShape** - Availability: 3.5.0 - requires SFCGAL >= 1.4.1. Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.
- **CG_OptimalConvexPartition** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Computes an optimal convex partition of the polygon geometry
- **CG_Orientation** - Availability: 3.5.0 检查几何体是否是实心体 (orientation) 检查几何体是否是实心体。
- **CG_StraightSkeleton** - Availability: 3.5.0 检查几何体是否是实心体 (straight skeleton) 检查几何体是否是实心体。
- **CG_Tessellate** - Availability: 3.5.0 检查几何体是否是实心体 (tessellation) 检查几何体是否是实心体 TIN 检查几何体是否是实心体。
- **CG_Triangulate** - Availability: 3.5.0 Triangulates a polygonal geometry
- **CG_Union** - Availability: 3.5.0 Computes the union of two geometries
- **CG_Visibility** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Compute a visibility polygon from a point or a segment in a polygon geometry
- **CG_Volume** - Availability: 3.5.0 Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **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

13.12.4 PostGIS Functions new or enhanced in 3.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.4

- **PostGIS_GEOS_Compiled_Version** - Availability: 3.4.0 Returns the version number of the GEOS library against which PostGIS was built.
- **PostGIS_PROJ_Compiled_Version** - Availability: 3.5.0 Returns the version number of the PROJ library against which PostGIS was built.
- **RenameTopoGeometryColumn** - Availability: 3.4.0 Renames a topogeometry column
- **RenameTopology** - Availability: 3.4.0 Renames a topology
- **ST_ClusterIntersectingWin** - Availability: 3.4.0 Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
- **ST_ClusterWithinWin** - Availability: 3.4.0 Window function that returns a cluster id for each input geometry, clustering using separation distance.
- **ST_CoverageInvalidEdges** - Availability: 3.4.0 Window function that finds locations where polygons fail to form a valid coverage.
- **ST_CoverageSimplify** - Availability: 3.4.0 Window function that simplifies the edges of a polygonal coverage.
- **ST_CoverageUnion** - Availability: 3.4.0 - requires GEOS >= 3.8.0 Computes the union of a set of polygons forming a coverage by removing shared edges.
- **ST_InverseTransformPipeline** - Availability: 3.4.0 Return a new geometry with coordinates transformed to a different spatial reference system using the inverse of a defined coordinate transformation pipeline.
- **ST_LargestEmptyCircle** - Availability: 3.4.0. Computes the largest circle not overlapping a geometry.
- **ST_LineExtend** - Availability: 3.4.0 Returns a line extended forwards and backwards by specified distances.
- **ST_TransformPipeline** - Availability: 3.4.0 Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.
- **TopoElement** - Availability: 3.4.0 Converts a topogeometry to a topoelement.
- **postgis_srs** - Availability: 3.4.0 Return a metadata record for the requested authority and srid.
- **postgis_srs_all** - Availability: 3.4.0 Return metadata records for every spatial reference system in the underlying Proj database.
- **postgis_srs_codes** - Availability: 3.4.0 Return the list of SRS codes associated with the given authority.
- **postgis_srs_search** - Availability: 3.4.0 Return metadata records for projected coordinate systems that have areas of usage that fully contain the bounds parameter.

Functions enhanced in PostGIS 3.4

- **PostGIS_Full_Version** - Enhanced: 3.4.0 now includes extra PROJ configurations NETWORK_ENABLED, URL_ENDPOINT and DATABASE_PATH of proj.db location Reports full PostGIS version and build configuration infos.
- **PostGIS_PROJ_Version** - Enhanced: 3.4.0 now includes NETWORK_ENABLED, URL_ENDPOINT and DATABASE_PATH of proj.db location Returns the version number of the PROJ4 library.
- **ST_AsSVG** - Enhanced: 3.4.0 to support all curve types Returns SVG path data for a geometry.
- **ST_ClosestPoint** - Enhanced: 3.4.0 - Support for geography. Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.
- **ST_LineSubstring** - Enhanced: 3.4.0 - Support for geography was introduced. Returns the part of a line between two fractional locations.
- **ST_Project** - Enhanced: 3.4.0 Allow geometry arguments and two-point form omitting azimuth. Returns a point projected from a start point by a distance and bearing (azimuth).
- **ST_Resample** - Enhanced: 3.4.0 max and min resampling options added `ST_Resample(geometry, geometry, 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. `ST_ShortestLine(geography, geography)`.

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.

13.12.5 PostGIS Functions new or enhanced in 3.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.3

- **RemoveUnusedPrimitives** - Availability: 3.3.0 Removes topology primitives which not needed to define existing TopoGeometry objects.
- **ST_3DConvexHull** - Availability: 3.3.0 `ST_3DConvexHull(geometry)`.
- **ST_3DUnion** - Availability: 3.3.0 aggregate variant was added Perform 3D union.
- **ST_AsMARC21** - Availability: 3.3.0 Returns geometry as a MARC21/XML record with a geographic datafield (034).
- **ST_GeomFromMARC21** - Availability: 3.3.0, requires libxml2 2.6+ Takes MARC21/XML geographic data as input and returns a PostGIS geometry object.
- **ST_Letters** - Availability: 3.3.0 Returns the input letters rendered as geometry with a default start position at the origin and default text height of 100.
- **ST_OptimalAlphaShape** - Availability: 3.3.0 - requires SFCGAL >= 1.4.1. Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.

- **ST_SimplifyPolygonHull** - Availability: 3.3.0. Computes a simplified topology-preserving outer or inner hull of a polygonal geometry.
- **ST_TriangulatePolygon** - Availability: 3.3.0. Computes the constrained Delaunay triangulation of polygons
- **postgis_sfcgal_full_version** - Availability: 3.3.0 Returns the full version of SFCGAL in use including CGAL and Boost versions

Functions enhanced in PostGIS 3.3

- **ST_ConcaveHull** - Enhanced: 3.3.0, GEOS native implementation enabled for GEOS 3.11+ Computes a possibly concave geometry that contains all input geometry vertices
- **ST_LineMerge** - Enhanced: 3.3.0 accept a directed parameter. Return the lines formed by sewing together a MultiLineString.

Functions changed in PostGIS 3.3

- **PostGIS_Extensions_Upgrade** - Changed: 3.3.0 support for upgrades from any PostGIS version. Does not work on all systems. Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.

13.12.6 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 新特性.

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** - 新特性: 2.1.0 新特性 C 新特性. 新特性 (新特性) 新特性.
- **ST_PixelAsCentroids** - 新特性: 2.1.0 新特性 C 新特性. 新特性 (新特性) 新特性 X, Y 新特性. 新特性.
- **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, CircularLineString, or NURBS-Curve.
- **ST_Value** - 新特性: 2.1.0 新特性 exclude nodata value 新特性. 新特性 columnx, rowy 新特性, 新特性. 新特性 1 新特性, 新特性 1 新特性. exclude_nodata_value 新特性, nodata 新特性. exclude_nodata_value 新特性, 新特性.
- **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 新特性.
- **ValidateTopology** - Changed: 3.2.0 added optional bbox parameter, perform face labeling and edge linking checks. Returns a set of validate_topology_returntype objects detailing issues with topology.

13.12.7 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 返回一个六边形，使用提供的边长和单元坐标在六边形网格空间内。
- **ST_HexagonGrid** - 2.1.0 返回一组六边形和单元索引，它们完全覆盖几何参数的边界。
- **ST_MaximumInscribedCircle** - 可用性: 3.1.0。返回一个最大内切圆。
- **ST_ReducePrecision** - 可用性: 3.1.0。返回一个有效的几何体，其点被舍入到网格容差。
- **ST_Square** - 2.1.0 返回一个正方形，使用提供的边长和单元坐标在正方形网格空间内。
- **ST_SquareGrid** - 2.1.0 返回一组网格正方形和单元索引，它们完全覆盖几何参数的边界。

Functions enhanced in PostGIS 3.1

- **ST_AsEWKT** - 增强: 3.1.0 支持可选精度参数。WKT(Well-Known Text) 格式，SRID 指定。
- **ST_ClusterKMeans** - 增强: 3.1.0 支持 3D 几何体和权重窗口函数，返回每个输入几何体使用 K-均值算法的簇 ID。
- **ST_Difference** - 增强: 3.1.0 接受 gridSize 参数。计算几何体 A 中不与几何体 B 相交的部分。
- **ST_Intersection** - 增强: 3.1.0 接受 gridSize 参数。计算几何体 A 和 B 的共享部分。
- **ST_MakeValid** - 增强: 3.1.0，添加移除带有 NaN 值的坐标。尝试使无效的几何体有效而不丢失顶点。
- **ST_Subdivide** - 增强: 3.1.0 接受 gridSize 参数。计算几何体的矩形细分。
- **ST_SymDifference** - 增强: 3.1.0 接受 gridSize 参数。计算几何体 A 和 B 的不相交部分。
- **ST_TileEnvelope** - 2.0.0 返回一个矩形多边形，在 Web Mercator (SRID:3857) 使用 XYZ 瓦片系统。
- **ST_UnaryUnion** - 增强: 3.1.0 接受 gridSize 参数。计算单个几何体的并集。
- **ST_Union** - 增强: 3.1.0 接受 gridSize 参数。计算输入几何体的点集并集。

Functions changed in PostGIS 3.1

- **ST_Count** - 2.2.0 返回 ST_Count(rastertable, rastercolumn, ...) 的结果。exclude_nodata_value 参数，NODATA 值。

- **ST_Force3D** - Changed: 3.1.0. Added support for supplying a non-zero Z value. `ST_Force3DXYZ(geometry, z_value)`. `ST_Force3DZ(geometry, z_value)`.
- **ST_Force3DM** - Changed: 3.1.0. Added support for supplying a non-zero M value. `ST_Force3DMXYZM(geometry, z_value, m_value)`.
- **ST_Force3DZ** - Changed: 3.1.0. Added support for supplying a non-zero Z value. `ST_Force3DXYZ(geometry, z_value)`.
- **ST_Force4D** - Changed: 3.1.0. Added support for supplying non-zero Z and M values. `ST_Force4DXYZM(geometry, z_value, m_value)`.
- **ST_Histogram** - Changed: 3.1.0 Removed `ST_Histogram(table_name, column_name)` variant. `ST_Histogram(bin; geometry, column_name)`. `ST_Histogram(bin; geometry, column_name, population)`. `ST_Histogram(bin; geometry, column_name, population, quantile)`. `ST_Histogram(bin; geometry, column_name, population, quantile, percentile)`.
- **ST_Quantile** - Changed: 3.1.0 Removed `ST_Quantile(table_name, column_name)` variant. `ST_Quantile(geometry, column_name, population)` (population) `ST_Quantile(geometry, column_name, quantile)` (quantile) `ST_Quantile(geometry, column_name, percentile)` (percentile).
- **ST_SummaryStats** - 2.2.0 `ST_SummaryStats(rastertable, rastercolumn, ...)` `ST_SummaryStats(rastertable, rastercolumn, band)`. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed if no band is specified.

13.12.8 PostGIS Functions new or enhanced in 3.0

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.0

- **CG_ConstrainedDelaunayTriangles** - 2.1.0 `CG_ConstrainedDelaunayTriangles(geometry)`. Return a constrained Delaunay triangulation around the given input geometry.
- **ST_3DLineInterpolatePoint** - 2.1.0 `ST_3DLineInterpolatePoint(geometry, fraction)`. Returns a point interpolated along a 3D line at a fractional location.
- **ST_ConstrainedDelaunayTriangles** - 2.1.0 `ST_ConstrainedDelaunayTriangles(geometry)`. Return a constrained Delaunay triangulation around the given input geometry.
- **ST_TileEnvelope** - 2.1.0 `ST_TileEnvelope(xmin, xmax, ymin, ymax, srid)`. 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 `PostGIS`.
- **ST_LocateBetween** - Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. Returns the portions of a geometry that match a measure range.
- **ST_LocateBetweenElevations** - Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. Returns the portions of a geometry that lie in an elevation (Z) range.
- **ST_Overlaps** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other
- **ST_Relate** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix
- **ST_Segmentize** - Enhanced: 3.0.0 Segmentize geometry now produces equal-length subsegments Returns a modified geometry/geography having no segment longer than a given distance.
- **ST_Touches** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have at least one point in common, but their interiors do not intersect
- **ST_Within** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of A lies in B, and their interiors have a point in common

Functions changed in PostGIS 3.0

- **PostGIS_Extensions_Upgrade** - Changed: 3.0.0 to repack loose extensions and support postgis_raster. Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.
- **ST_3DDistance** - Changed: 3.0.0 - SFCGAL version removed `(SRS 3)`.
- **ST_3DIntersects** - Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs. Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- **ST_Area** - Changed: 3.0.0 - does not depend on SFCGAL anymore.
- **ST_AsGeoJSON** - Changed: 3.0.0 support records as input Return a geometry or feature in GeoJSON format.
- **ST_AsGeoJSON** - Changed: 3.0.0 output SRID if not EPSG:4326. Return a geometry or feature in GeoJSON format.
- **ST_AsKML** - Changed: 3.0.0 - Removed the "versioned" variant signature `GML 2 GML 3`.

- **ST_Distance** - Changed: 3.0.0 - does not depend on SFCGAL anymore. `ST_Distance` 3 `ST_Distance` (longest) `ST_Distance`.
- **ST_Intersection** - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the shared portion of geometries A and B.
- **ST_Intersects** - Changed: 3.0.0 SFCGAL version removed and native support for 2D TINS added. Tests if two geometries intersect (they have at least one point in common)
- **ST_Union** - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the point-set union of the input geometries.

13.12.9 PostGIS Functions new or enhanced in 2.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.5

- **PostGIS_Extensions_Upgrade** - Availability: 2.5.0 Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.
- **ST_Angle** - Availability: 2.5.0 Returns the angle in degrees between two line segments. (longest) `geometry, geometry`.
- **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-band raster from the source raster and specified bands representing Red, Green and Blue
- **ST_LineInterpolatePoints** - Availability: 2.5.0 Returns points interpolated along a line at a fractional interval.
- **ST_OrientedEnvelope** - Availability: 2.5.0. Returns a minimum-area rectangle containing a geometry.
- **ST_QuantizeCoordinates** - Availability: 2.5.0 Sets least significant bits of coordinates to zero
- **ST_RastFromHexWKB** - Availability: 2.5.0 Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.
- **ST_RastFromWKB** - Availability: 2.5.0 Return a raster value from a Well-Known Binary (WKB) raster.
- **ST_SetBandIndex** - Availability: 2.5.0 Update the external band number of an out-db band
- **ST_SetBandPath** - Availability: 2.5.0 Update the external path and band number of an out-db band

Functions enhanced in PostGIS 2.5

- **ST_AsBinary/ST_AsWKB** - Enhanced: 2.5.0 Addition of ST_AsWKB Return the Well-Known Binary (WKB) representation of the raster.

- **ST_AsMVT** - Enhanced: 2.5.0 - added support parallel query. Aggregate function returning a MVT representation of a set of rows.
- **ST_AsText** - Enhanced: 2.5 - optional parameter precision introduced. `WKT(Well-Known Text) SRID`.
- **ST_BandMetaData** - Enhanced: 2.5.0 to include `outdbbandnum`, `filesize` and `filetimestamp` for outdb rasters. `1`.
- **ST_Buffer** - Enhanced: 2.5.0 - `ST_Buffer` geometry support was enhanced to allow for side buffering specification `side=both|left|right`. Computes a geometry covering all points within a given distance from a geometry.
- **ST_GeomFromGeoJSON** - Enhanced: 2.5.0 can now accept `json` and `jsonb` as inputs. GeoJSON PostGIS.
- **ST_GeometricMedian** - Enhanced: 2.5.0 Added support for `M` as weight of points. (median).
- **ST_Intersects** - Enhanced: 2.5.0 Supports `GEOMETRYCOLLECTION`. Tests if two geometries intersect (they have at least one point in common)
- **ST_OffsetCurve** - Enhanced: 2.5 - added support for `GEOMETRYCOLLECTION` and `MULTILINESTRING`. Returns an offset line at a given distance and side from an input line.
- **ST_Scale** - Enhanced: 2.5.0 support for scaling relative to a local origin (origin parameter) was introduced. Scales a geometry by given factors.
- **ST_Split** - Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced. Returns a collection of geometries created by splitting a geometry by another geometry.
- **ST_Subdivide** - Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5. Computes a rectilinear subdivision of a geometry.

Functions changed in PostGIS 2.5

- **ST_GDALDrivers** - Changed: 2.5.0 - add `can_read` and `can_write` columns. Returns a list of raster formats supported by PostGIS through GDAL. Only those formats with `can_write=True` can be used by `ST_AsGDALRaster`

13.12.10 PostGIS Functions new or enhanced in 2.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.4

- **ST_AsGeobuf** - 2.2.0. Return a Geobuf representation of a set of rows.
- **ST_AsMVT** - 2.2.0. Aggregate function returning a MVT representation of a set of rows.
- **ST_AsMVTGeom** - 2.2.0. Transforms a geometry into the coordinate space of a MVT tile.
- **ST_Centroid** - Availability: 2.4.0 support for geography was introduced.
- **ST_ForcePolygonCCW** - 2.2.0. Orients all exterior rings counter-clockwise and all interior rings clockwise.

- **ST_ForcePolygonCW** - 2.2.0 新新新新新新新新新新. Orients all exterior rings clockwise and all interior rings counter-clockwise.
- **ST_FrechetDistance** - Availability: 2.4.0 - requires GEOS >= 3.7.0 新新新新新 3 新新新 (shortest) 新新新新新新新新.
- **ST_IsPolygonCCW** - 2.2.0 新新新新新新新新新新. Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.
- **ST_IsPolygonCW** - 2.2.0 新新新新新新新新新新. Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.
- **ST_MakeEmptyCoverage** - 2.2.0 新新新新新新新新新新. Cover georeferenced area with a grid of empty raster tiles.

Functions enhanced in PostGIS 2.4

- **ST_AsTWKB** - Enhanced: 2.4.0 memory and speed improvements. 新新 TWKB(Tiny Well-Known Binary) 新新新新新.
- **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. 新新新新新新新新新新新新新新新新新新.

Functions changed in PostGIS 2.4

- **=** - Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use `ST_Equals` 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.

13.12.11 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. 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. 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. 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. 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. 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. 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. 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. 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. 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 `Populate_Topology_Layer`. Adds missing entries to topology.layer table by reading metadata from topo tables.
- **ST_ClusterDBSCAN** - 2.3.0 `ST_ClusterDBSCAN`. Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- **ST_ClusterKMeans** - 2.3.0 `ST_ClusterKMeans`. Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST_GeneratePoints** - 2.3.0 `ST_GeneratePoints`. Generates a multipoint of random points contained in a Polygon or MultiPolygon.
- **ST_GeometricMedian** - 2.3.0 `ST_GeometricMedian`. `ST_GeometricMedian` (median) `ST_GeometricMedian`.
- **ST_MakeLine** - 2.0.0 `ST_MakeLine`. `ST_MakeLine`, `ST_MakeLine`.
- **ST_MinimumBoundingRadius** - 2.3.0 `ST_MinimumBoundingRadius`. Returns the center point and radius of the smallest circle that contains a geometry.
- **ST_MinimumClearance** - 2.3.0 `ST_MinimumClearance`. `ST_MinimumClearance` (robustness) `ST_MinimumClearance` (clearance) `ST_MinimumClearance`.
- **ST_MinimumClearanceLine** - 2.3.0 `ST_MinimumClearanceLine`. GEOS 3.6.0 `ST_MinimumClearanceLine`. `ST_MinimumClearanceLine` 2 `ST_MinimumClearanceLine`, `ST_MinimumClearanceLine`.
- **ST_Normalize** - 2.3.0 `ST_Normalize`. `ST_Normalize`.
- **ST_Points** - 2.3.0 `ST_Points`. `ST_Points`.
- **ST_VoronoiLines** - 2.3.0 `ST_VoronoiLines`. Returns the boundaries of the Voronoi diagram of the vertices of a geometry.
- **ST_VoronoiPolygons** - 2.3.0 `ST_VoronoiPolygons`. Returns the cells of the Voronoi diagram of the vertices of a geometry.
- **ST_WrapX** - Availability: 2.3.0 requires GEOS X `ST_WrapX`.
- **TopoGeom_addElement** - 2.3 `TopoGeom_addElement`. Adds an element to the definition of a TopoGeometry.

- **TopoGeom_remElement** - 2.3 国国国国国国国国国国国国. Removes an element from the definition of a TopoGeometry.
- **~(box2df,box2df)** - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- **~(geometry,box2df)** - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. 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** - 国国国国: 2.3.0 国国国国国国国国国国 (-1 国国国国国国国国) 国国国国国国国国国国. ST_LineString 国国 ST_CircularString 国国国国国国国国国国国国国国国国.

13.12.12 PostGIS Functions new or enhanced in 2.2

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.2

- **<<->>** - Availability: 2.2.0. Returns the n-D distance between the A and B geometries or bounding boxes
- **ST_3DDifference** - 2.2.0 国国国国国国国国国国国国. 3 国国国国国国国国国国.
- **ST_3DUnion** - 2.2.0 国国国国国国国国国国国国. Perform 3D union.
- **ST_ApproximateMedialAxis** - 2.2.0 国国国国国国国国国国国国. 国国国国国国国国国国国国国国国国.

- **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。创建几何体的概述版本。
- **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。将几何体转换为实心体。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。对几何体进行重新分块。
- **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. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a set of raster. Band 1 is assumed if no band is specified.
- **ST_SwapOrdinates** - 2.2.0 新增。交换几何体的两个维度的坐标。
- **ST_Volume** - 2.2.0 新增。Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **postgis.enable_outdb_rasters** - 2.2.0 新增。DB 外部存储的栅格数据。
- **postgis.gdal_datapath** - 2.2.0 新增。GDAL 的 GDAL_DATA 目录路径。
- **postgis.gdal_enabled_drivers** - 2.2.0 新增。PostGIS 支持的 GDAL 驱动程序。
- **|=** - Availability: 2.2.0. A 与 B 的最接近点 (closest point of approach) 和轨迹 (trajectory)。

Functions enhanced in PostGIS 2.2

- **<->** - Enhanced: 2.2.0 -- True KNN ("K nearest neighbor") behavior for geometry and geography. Note for geography KNN is based on sphere rather than spheroid. A 与 B 的 2 个最近点。
- **AsTopoJSON** - 新增: 2.2.1 新增 (puntal) TopoGeometry 到 TopoJSON。
- **ST_Area** - 新增: 2.2.0 新增。GeographicLib 4.2.0 版本。Proj 4.9.0 版本。
- **ST_AsX3D** - 新增: 2.2.0 新增 (x/y, z) 坐标。ISO-IEC-19776-1.2-X3DEncodings-XML。
- **ST_Azimuth** - 新增: 2.2.0 新增。GeographicLib 4.2.0 版本。Proj 4.9.0 版本。
- **ST_Distance** - 新增: 2.2.0 新增。GeographicLib 4.2.0 版本。Proj 4.9.0 版本。
- **ST_Scale** - Enhanced: 2.2.0 support for scaling all dimension (factor parameter) was introduced. Scales a geometry by given factors.
- **ST_Split** - Enhanced: 2.2.0 support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced. Returns a collection of geometries created by splitting a geometry by another geometry.
- **ST_Summary** - 新增: 2.2.0 新增 TIN (curve) 数据。

Functions changed in PostGIS 2.2

- **<->** - Changed: 2.2.0 -- Old hybrid-syntax workarounds may be slower once true KNN is available. See examples below. A 与 B 的 2 个最近点。

- ### 13.12.13 PostGIS Functions new or enhanced in 2.1

Functions new in PostGIS 2.1

- **=** - 2.1.0 `bool`. A `bool` B `bool` `TRUE` `bool`. `bool` `bool`.
- **AsTopoJSON** - 2.1.0 `TopoGeometry` `TopoJSON` `TopoJSON`.
- **ST_3DArea** - 2.1.0 `float`. 3 `float`. `float` 0 `float` `float`.
- **ST_3DIntersection** - 2.1.0 `float`. 3 `float`.
- **ST_Box2dFromGeoHash** - 2.1.0 `float`. GeoHash `float` BOX2D `float` `float`.
- **ST_ColorMap** - 2.1.0 `float`. `float` 8BUI `float` (grayscale, RGB, RGBA) `float` 4 `float`. `float` 1 `float`.

- **ST_Contains** - 2.1.0 返回布尔值。如果 rastA 包含 rastB 的几何体，则返回 true，否则返回 false。
- **ST_ContainsProperly** - 2.1.0 返回布尔值。如果 rastB 完全包含在 rastA 中，且 rastA 的面积大于 rastB 的面积，则返回 true。
- **ST_CoveredBy** - 2.1.0 返回布尔值。如果 rastA 的几何体被 rastB 的几何体完全覆盖，则返回 true。
- **ST_Covers** - 2.1.0 返回布尔值。如果 rastB 的几何体完全覆盖 rastA 的几何体，则返回 true。
- **ST_DFullyWithin** - 2.1.0 返回布尔值。如果 rastA 的几何体完全在 rastB 的几何体内部，且距离不超过指定距离，则返回 true。
- **ST_DWithin** - 2.1.0 返回布尔值。如果 rastA 的几何体与 rastB 的几何体之间的距离不超过指定距离，则返回 true。
- **ST_DelaunayTriangles** - 2.1.0 返回几何体。Returns the Delaunay triangulation of the vertices of a geometry.
- **ST_Disjoint** - 2.1.0 返回布尔值。如果 rastA 和 rastB 的几何体没有公共点，则返回 true。
- **ST_DumpValues** - 2.1.0 返回文本。返回 rast 的 2 个值。
- **ST_Extrude** - 2.1.0 返回几何体。将 rast 的几何体沿 z 轴拉伸指定距离。
- **ST_ForceLHR** - 2.1.0 返回几何体。LHR(Left Hand Reverse; 左手) 规则。
- **ST_FromGDALRaster** - 2.1.0 返回几何体。从 GDAL 栅格创建几何体。
- **ST_GeomFromGeoHash** - 2.1.0 返回几何体。GeoHash 字符串。
- **ST_InvDistWeight4ma** - 2.1.0 返回几何体。返回 rast 的 4 个最近邻的权重。
- **ST_MapAlgebra (callback function version)** - 2.1.0 返回几何体。使用回调函数对 rast 进行映射代数操作。
- **ST_MapAlgebra (expression version)** - 2.1.0 返回几何体。使用表达式对 rast 进行映射代数操作。
- **ST_MinConvexHull** - 2.1.0 返回几何体。返回 rast 的最小凸包。
- **ST_MinDist4ma** - 2.1.0 返回几何体。返回 rast 的 4 个最近邻的最小距离。
- **ST_MinkowskiSum** - 2.1.0 返回几何体。返回 rast 的 Minkowski 和。
- **ST_NearestValue** - 2.1.0 返回几何体。columnx 和 rowy 指定栅格，返回 rast 的最近值。
- **ST_Neighborhood** - 2.1.0 返回几何体。columnx 和 rowy 指定栅格，返回 rast 的邻域。
- **ST_NotSameAlignmentReason** - 2.1.0 返回文本。返回 rast 的不同对齐原因。
- **ST_Orientation** - 2.1.0 返回文本。返回 rast 的方向 (orientation)。

- **ST_Overlaps** - 2.1.0 返回布尔值。如果 rastA 与 rastB 重叠，则返回 true，否则返回 false。
- **ST_PixelAsCentroid** - 2.1.0 返回浮点数组。返回指定像素 (row, col) 的质心。
- **ST_PixelAsCentroids** - 2.1.0 返回浮点数组。返回指定像素 (row, col) 的质心 X, Y 坐标。
- **ST_PixelAsPoint** - 2.1.0 返回浮点数组。返回指定像素 (row, col) 的点。
- **ST_PixelAsPoints** - 2.1.0 返回浮点数组。返回指定像素 (row, col) 的点 X, Y 坐标。
- **ST_PixelOfValue** - 2.1.0 返回浮点数组。返回指定像素 (row, col) 的值 columnx, rowy 的像素。
- **ST_PointFromGeoHash** - 2.1.0 返回浮点数组。GeoHash 字符串转换为点。
- **ST_RasterToWorldCoord** - 2.1.0 返回浮点数组。返回指定像素 (row, col) 的世界坐标 X, Y (经度, 纬度)。
- **ST_Resize** - 2.1.0 返回浮点数组。GDAL 1.6.1 的 Resize 函数。
- **ST_Roughness** - 2.1.0 返回浮点数组。DEM 的粗糙度 (roughness) 值。
- **ST_SetValues** - 2.1.0 返回浮点数组。设置指定像素 (row, col) 的值。
- **ST_Simplify** - 2.1.0 返回浮点数组。使用 Douglas-Peucker 算法简化 TopoGeometry。
- **ST_StraightSkeleton** - 2.1.0 返回浮点数组。返回直骨架 (straight skeleton)。
- **ST_Summary** - 2.1.0 返回浮点数组。返回摘要信息。
- **ST_TPI** - 2.1.0 返回浮点数组。返回地形位置指数 (Topographic Position Index)。
- **ST_TRI** - 2.1.0 返回浮点数组。返回地形粗糙度指数 (Terrain Ruggedness Index)。
- **ST_Tessellate** - 2.1.0 返回浮点数组。返回三角网 (tessellation)。
- **ST_Tile** - 2.1.0 返回浮点数组。返回瓦片 (tile)。
- **ST_Touches** - 2.1.0 返回布尔值。如果 rastA 与 rastB 接触，则返回 true，否则返回 false。
- **ST_Union** - 2.1.0 返回浮点数组。返回 ST_Union(rast, unionarg) 的结果。
- **ST_Within** - 2.1.0 返回布尔值。如果 rastB 在 rastA 内部，则返回 true，否则返回 false。
- **ST_WorldToRasterCoord** - 2.1.0 返回浮点数组。返回指定 X, Y (经度, 纬度) 的像素 (row, col)。
- **UpdateRasterSRID** - 2.1.0 返回浮点数组。更新 Raster 的 SRID。

- **clearTopoGeom** - 2.1 清除拓扑几何的内容。
- **postgis_sfcgal_version** - 2.1.0 返回 SFCGAL 版本。

Functions enhanced in PostGIS 2.1

- **ST_AddBand** - 2.1.0 新增 addbandarg 参数。用于在现有栅格中添加新波段。
- **ST_AddBand** - 2.1.0 新增 DB 支持。用于在数据库中添加新波段。
- **ST_AsBinary/ST_AsWKB** - 2.1.0 新增 outasin 参数。返回 Well-Known Binary (WKB) 表示的栅格。
- **ST_AsGML** - 2.1.0 新增 GML 3 ID 支持。返回 GML 2 或 GML 3 表示的栅格。
- **ST_Aspect** - 2.1.0 新增 ST_MapAlgebra() 参数，支持 interpolate_nodata 选项。
- **ST_Boundary** - 2.1.0 新增支持。返回栅格的边界。
- **ST_Clip** - 2.1.0 新增 C 参数。返回被输入几何裁剪的栅格。
- **ST_DWithin** - 增强：2.1.0 改进地理学速度。参见 Making Geography faster for details。测试两个几何是否在给定距离内。
- **ST_DWithin** - 增强：2.1.0 支持曲线几何。测试两个几何是否在给定距离内。
- **ST_Distance** - 增强：2.1.0 改进地理学速度。参见 Making Geography faster for details。返回 3 个最长距离。
- **ST_Distance** - 2.1.0 新增支持。返回 3 个最长距离。
- **ST_Distinct4ma** - 2.1.0 新增支持。返回 2 个最长距离。
- **ST_DumpPoints** - 增强：2.1.0 更快的速度。重新实现为原生 C。
- **ST_HillShade** - 2.1.0 新增 ST_MapAlgebra() 参数，支持 interpolate_nodata 选项。
- **ST_MakeValid** - 增强：2.1.0，新增对 GEOMETRYCOLLECTION 和 MULTIPOINT 的支持。尝试使无效几何有效而不丢失顶点。
- **ST_Max4ma** - 2.1.0 新增支持。返回 2 个最长距离。
- **ST_Mean4ma** - 2.1.0 新增支持。返回 2 个最长距离。

- **ST_Min4ma** - 函数: 2.1.0 函数返回 2 个最小值。返回两个最小值的平均值。
- **ST_PixelAsPolygons** - 函数: 2.1.0 函数返回 exclude_nodata_value 是否排除 NODATA 值。返回 X, Y 坐标和像素值。
- **ST_Polygon** - 函数: 2.1.0 函数返回多边形 (C 语言)。返回 NODATA 值。
- **ST_Range4ma** - 函数: 2.1.0 函数返回 2 个范围。返回两个范围的平均值。
- **ST_SameAlignment** - 函数: 2.1.0 函数返回是否相同。返回 true, false, 或 null (如果输入是 null)。
- **ST_Segmentize** - 函数: 2.1.0 函数返回分段。Returns a modified geometry/geography having no segment longer than a given distance.
- **ST_SetGeoReference** - 函数: 2.1.0 函数 ST_SetGeoReference(raster, double precision, ...) 设置地理参考。返回 6 个参数。GDAL 或 ESRI 格式。返回 GDAL 格式。
- **ST_SetValue** - 函数: 2.1.0 函数 ST_SetValue() 设置值。返回 ST_SetValues() 的 geomval[] (wrapper) 函数。返回 columnx, rowy 值。返回 1 个值, 返回 1 个值。
- **ST_Slope** - 函数: 2.1.0 函数 ST_MapAlgebra() 函数, 返回 units, scale, interpolate_nodata 值。返回 (函数) 值。返回 true。
- **ST_StdDev4ma** - 函数: 2.1.0 函数返回 2 个标准差。返回两个标准差的平均值。
- **ST_Sum4ma** - 函数: 2.1.0 函数返回 2 个和。返回两个和的平均值。
- **ST_Summary** - 函数: 2.1.0 函数。返回 S 值。
- **ST_Transform** - 函数: 2.1.0 函数 ST_Transform(rast, alignto) 函数。返回 NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos 函数。返回 NearestNeighbor 函数。
- **ST_Union** - 函数: 2.1.0 函数返回多边形 (C 语言)。返回 1 个值。
- **ST_Union** - 函数: 2.1.0 函数 ST_Union(rast) 返回 1 个值。PostGIS 函数。返回 1 个值。
- **ST_Union** - 函数: 2.1.0 函数 ST_Union(rast, uniontype) 返回 4 个值。返回 1 个值。
- **toTopoGeom** - 函数: 2.1.0 函数 TopoGeometry 函数。Converts a simple Geometry into a topo geometry.

Functions changed in PostGIS 2.1

- **ST_Aspect** - 新增: 2.1.0 新增返回栅格数据的坡度。2.1.0 新增返回栅格数据的坡向。返回栅格数据的坡度 (以度为单位) 和坡向。返回栅格数据的坡度。
- **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, 在 2.0.x 中 ST_Force_2D 被弃用。返回 2D 强制视图。
- **ST_Force3D** - 新增: 2.1.0 新增, 在 2.0.x 中 ST_Force_3D 被弃用。返回 XYZ 强制视图。ST_Force3DZ 被弃用。
- **ST_Force3DM** - 新增: 2.1.0 新增, 在 2.0.x 中 ST_Force_3DM 被弃用。返回 XYM 强制视图。
- **ST_Force3DZ** - 新增: 2.1.0 新增, 在 2.0.x 中 ST_Force_3DZ 被弃用。返回 XYZ 强制视图。
- **ST_Force4D** - 新增: 2.1.0 新增, 在 2.0.x 中 ST_Force_4D 被弃用。返回 XYZM 强制视图。
- **ST_ForceCollection** - 新增: 2.1.0 新增, 在 2.0.x 中 ST_Force_Collection 被弃用。返回强制集合视图。
- **ST_HillShade** - 新增: 2.1.0 新增返回栅格数据的阴影。2.1.0 新增返回栅格数据的阴影。返回栅格数据的阴影, 返回栅格数据的阴影, 返回栅格数据的阴影。
- **ST_LineInterpolatePoint** - 新增: 2.1.0 新增, 在 2.0.x 中 ST_Line_Interpolate_Point 被弃用。Returns a point interpolated along a line at a fractional location.
- **ST_LineLocatePoint** - 新增: 2.1.0 新增, 在 2.0.x 中 ST_Line_Locate_Point 被弃用。Returns the fractional location of the closest point on a line to a point.
- **ST_LineSubstring** - 新增: 2.1.0 新增, 在 2.0.x 中 ST_Line_Substring 被弃用。Returns the part of a line between two fractional locations.
- **ST_PixelAsCentroids** - 新增: 2.1.1 新增 exclude_nodata_value 参数。返回栅格数据的质心 (以度为单位) 的 X, Y 坐标。返回栅格数据的质心。
- **ST_PixelAsPoints** - 新增: 2.1.1 新增 exclude_nodata_value 参数。返回栅格数据的点 (以度为单位) 的 X, Y 坐标。返回栅格数据的点。
- **ST_PixelAsPolygons** - 新增: 2.1.1 新增 exclude_nodata_value 参数。返回栅格数据的多边形 (以度为单位) 的 X, Y 坐标。返回栅格数据的多边形。
- **ST_Polygon** - 新增: 2.1.0 新增返回栅格数据的多边形, 返回栅格数据的多边形。NODATA 返回栅格数据的多边形。
- **ST_RasterToWorldCoordX** - 新增: 2.1.0 新增 ST_Raster2WorldCoordX 函数。返回栅格数据的 X 坐标。返回栅格数据的 X 坐标。
- **ST_RasterToWorldCoordY** - 新增: 2.1.0 新增 ST_Raster2WorldCoordY 函数。返回栅格数据的 Y 坐标。返回栅格数据的 Y 坐标。
- **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.

- ### 13.12.14 PostGIS Functions new or enhanced in 2.0

Functions new in PostGIS 2.0

- **&&** - 2.0.0 `[[[A, B]]]`. A `[[[A, B]]]` B `[[[A, B]]]` TRUE `[[[A, B]]]`.
- **&&&** - 2.0.0 `[[[A, B]]]`. A `[[[A, B]]]` B `[[[A, B]]]` TRUE `[[[A, B]]]`.
- **<#>** - Availability: 2.0.0. A `[[[A, B]]]` 2 `[[[A, B]]]`.
- **<->** - Availability: 2.0.0 -- Weak KNN provides nearest neighbors based on geometry centroid distances instead of true distances. Exact results for points, inexact for all other types. A `[[[A, B]]]` 2 `[[[A, B]]]`.
- **@** - 2.0.0 `[[[A, B]]]` raster @ raster, raster @ geometry `[[[A, B]]]`. B `[[[A, B]]]` A `[[[A, B]]]` TRUE `[[[A, B]]]`. `[[[A, B]]]`.
- **@** - 2.0.5 `[[[A, B]]]` geometry @ raster `[[[A, B]]]`. B `[[[A, B]]]` A `[[[A, B]]]` TRUE `[[[A, B]]]`. `[[[A, B]]]`.
- **AddEdge** - 2.0.0 `[[[A, B]]]`. `[[[A, B]]]` (edgeid) `[[[A, B]]]` ID(edgeid) `[[[A, B]]]`.
- **AddFace** - 2.0.0 `[[[A, B]]]`. `[[[A, B]]]` (face primitive) `[[[A, B]]]` `[[[A, B]]]`.
- **AddNode** - 2.0.0 `[[[A, B]]]`. `[[[A, B]]]` ID(nodeid) `[[[A, B]]]`. `[[[A, B]]]` ID `[[[A, B]]]`.
- **AddOverviewConstraints** - 2.0.0 `[[[A, B]]]`. `[[[A, B]]]` (overview) `[[[A, B]]]`.

- **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。移除 (overview) 约束。
- **DropRasterConstraints** - 2.0.0 新增。移除 PostGIS 2.0.0 引入的栅格约束。
- **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。获取指定节点的边。
- **GetRingEdges** - 2.0.0 新增。获取指定面的边界边。
- **GetTopoGeomElements** - 2.0.0 新增。Returns a set of topoelement objects containing the topological element_id,element_type of the given TopoGeometry (primitive elements).
- **GetTopologySRID** - 2.0.0 新增。获取 topology.topology 的 SRID。
- **Polygonize** - 2.0.0 新增。Finds and registers all faces defined by topology edges.
- **ST_3DClosestPoint** - 2.0.0 新增。g2 与 g1 的 3D 最近点。
- **ST_3DDFullyWithin** - 2.0.0 新增。Tests if two 3D geometries are entirely within a given 3D distance
- **ST_3DDWithin** - 2.0.0 新增。Tests if two 3D geometries are within a given 3D distance
- **ST_3DDistance** - 2.0.0 新增。计算两个 3D 几何体 (SRS 一致) 的 3D 距离。
- **ST_3DIntersects** - 2.0.0 新增。Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- **ST_3DLongestLine** - 2.0.0 新增。获取 3D 几何体 (longest) 的最长线。
- **ST_3DMaxDistance** - 2.0.0 新增。计算两个 3D 几何体 (SRS 一致) 的 3D 最大距离。
- **ST_3DShortestLine** - 2.0.0 新增。获取 3D 几何体 (shortest) 的最短线。
- **ST_AddEdgeModFace** - 2.0 新增。添加边并修改面。

- **ST_AddEdgeNewFaces** - 2.0.0 `geometry, geometry, integer`. Returns the new faces created by the operation. The faces are returned as a `geometry` object.
- **ST_AsGDALRaster** - 2.0.0 `geometry, text, integer`. GDAL 1.6.0 `geometry, text, integer`. 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_AsJPEG** - 2.0.0 `geometry, integer`. GDAL 1.6.0 `geometry, integer`. Returns the raster tile as a JPEG (Joint Photographic Experts Group) image (byte array). The image is returned as a `bytea` object. The image is 1 byte per pixel. The image is 3 bytes per pixel (RGB). The image is 3 bytes per pixel (RGB).
- **ST_AsLatLonText** - 2.0 `geometry, text, integer`. Returns the raster tile as a text representation of the geometry.
- **ST_AsPNG** - 2.0.0 `geometry, integer`. GDAL 1.6.0 `geometry, integer`. Returns the raster tile as a PNG (Portable Network Graphics) image (byte array). The image is returned as a `bytea` object. The image is 1 byte per pixel. The image is 3 bytes per pixel (RGB). The image is 4 bytes per pixel (RGBA). The image is 4 bytes per pixel (RGBA).
- **ST_AsRaster** - 2.0.0 `geometry, text, integer`. GDAL 1.6.0 `geometry, text, integer`. PostGIS `geometry, text, integer`. PostGIS `geometry, text, integer`.
- **ST_AsTIFF** - 2.0.0 `geometry, integer`. GDAL 1.6.0 `geometry, integer`. 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_AsX3D** - 2.0.0 `geometry, text, integer`. Returns the raster tile as an X3D XML document. The X3D XML document is returned as a `text` object.
- **ST_Aspect** - 2.0.0 `geometry, integer`. Returns the aspect of the raster tile. The aspect is returned as a `float` value.
- **ST_Band** - 2.0.0 `geometry, integer, integer`. Returns the raster tile as a `bytea` object. The raster tile is returned as a `bytea` object.
- **ST_BandIsNoData** - 2.0.0 `geometry, integer, integer`. Returns the raster tile as a `boolean` value. The raster tile is returned as a `boolean` value.
- **ST_Clip** - 2.0.0 `geometry, geometry, integer, boolean`. 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_CollectionHomogenize** - 2.0.0 `geometry, integer`. Returns the simplest representation of a geometry collection.
- **ST_ConcaveHull** - 2.0.0 `geometry, integer`. Computes a possibly concave geometry that contains all input geometry vertices.
- **ST_Count** - 2.0.0 `geometry, integer, integer, integer`. Returns the count of the raster tile. The count is returned as a `integer` value. The count is returned as a `integer` value. The count is returned as a `integer` value. The count is returned as a `integer` value.
- **ST_CreateTopoGeo** - 2.0 `geometry, integer`. Returns the raster tile as a `topology` object. The raster tile is returned as a `topology` object.
- **ST_Distinct4ma** - 2.0.0 `geometry, integer, integer, integer, integer`. Returns the raster tile as a `geometry` object. The raster tile is returned as a `geometry` object.
- **ST_FlipCoordinates** - 2.0.0 `geometry, integer`. Returns a version of a geometry with X and Y axis flipped.

- ST_GDALDrivers** - 2.0.0 `TEXT[]`. GDAL 1.6.0 `TEXT[]`. 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_GeomFromGeoJSON** - 2.0.0 `GEOMETRY`. JSON-C 0.9 `TEXT`. GeoJSON `TEXT` PostGIS `GEOMETRY`.
- ST_GetFaceEdges** - 2.0 `TEXT`. `aface TEXT[]` returns the faces of a geometry.
- ST_HasNoBand** - 2.0.0 `BOOLEAN`. Checks if a geometry has no band. If yes, returns 1, otherwise 0.
- ST_HillShade** - 2.0.0 `REAL`. Takes a geometry, a float, a text, and a text as input. Returns a real value.
- ST_Histogram** - 2.0.0 `TEXT`. Takes a geometry, a bin, and a text as input. Returns a text value.
- ST_InterpolatePoint** - 2.0.0 `POINT`. Takes a geometry, a float, and a text as input. Returns a point (M) value.
- ST_IsEmpty** - 2.0.0 `BOOLEAN`. Checks if a geometry is empty (width = 0, height = 0). Returns 1 or 0.
- ST_IsValidDetail** - 2.0.0 `TEXT`. Returns a valid_detail row stating if a geometry is valid or if not a reason and a location.
- ST_IsValidReason** - Availability: 2.0 version taking flags. Returns text stating if a geometry is valid, or a reason for invalidity.
- ST_MakeLine** - 2.0.0 `LINESTRING`. Takes two geometries as input. Returns a linestring.
- ST_MakeValid** - 2.0.0 `GEOMETRY`. Attempts to make an invalid geometry valid without losing vertices.
- ST_MapAlgebraExpr** - 2.0.0 `TEXT`. Takes 1 text: A PostgreSQL expression, a text, and 1 integer. Returns a text value.
- ST_MapAlgebraExpr** - 2.0.0 `TEXT`. Takes 2 texts: A PostgreSQL expression, a text, and 1 integer. Returns a text value. extenttype INTERSECTION, UNION, FIRST, SECOND.
- ST_MapAlgebraFct** - 2.0.0 `TEXT`. Takes 1 text: A PostgreSQL function name, a text, and 1 integer. Returns a text value.
- ST_MapAlgebraFct** - 2.0.0 `TEXT`. Takes 2 texts: A PostgreSQL function name, a text, and 1 integer. Returns a text value. extenttype INTERSECTION.
- ST_MapAlgebraFctNgb** - 2.0.0 `TEXT`. Takes 1 text: A PostgreSQL function name (Map Algebra Nearest Neighbor), a text, and 1 integer. Returns a text value.
- ST_Max4ma** - 2.0.0 `TEXT`. Takes a geometry and a text as input. Returns a text value.

- **ST_Mean4ma** - 2.0.0 返回多边形中所有点的平均坐标。返回一个点几何体。
- **ST_Min4ma** - 2.0.0 返回多边形中所有点的最低坐标。返回一个点几何体。
- **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。如果新点几何体已经存在，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，并用具有与第一个边相同方向的边替换它们。
- **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。如果输入为 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 返回多边形几何体或地理实体的周长。
- **ST_PixelAsPolygon** - 2.0.0 返回像素的多边形表示。
- **ST_PixelAsPolygons** - 2.0.0 返回像素的多边形表示。返回 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 返回给定人口（population）的指定分位数（quantile）的坐标。例如，25%、50%、75% 分位数（percentile）。
- **ST_Range4ma** - 2.0.0 返回多边形中所有点的范围。
- **ST_Reclass** - 2.0.0 根据 nband 重新分类多边形。nband 可以是 1 或更多。例如：16BUI 或 8BUI。
- **ST_RelateMatch** - 2.0.0 测试 DE-9IM 交集矩阵是否匹配指定的交集矩阵模式。
- **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其空间。
- **ST_RemEdgeNewFace** - 2.0 移除边，并创建新的面。
- **ST_Resample** - 2.0.0 使用 GDAL 1.6.1 重新采样线。返回重新采样的线。

- **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) 坐标进行重新采样。NearestNeighbor(最近邻), Bilinear, Cubic, CubicSpline 和 Lanczos 重新采样算法。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 (即是否都是左上角对齐或都是右下角对齐)。
- **ST_SetBandIsNoData** - 2.0.0 新增函数。设置 isnodata 标志。
- **ST_SharedPaths** - 2.0.0 新增函数。返回两个几何体共享的路径。
- **ST_Slope** - 2.0.0 新增函数。计算坡度 (以度为单位)。
- **ST_Snap** - 2.0.0 新增函数。将几何体 snap 到最近的栅格单元。
- **ST_SnapToGrid** - 2.0.0 新增函数。GDAL 1.6.1 版本。将几何体 snap 到最近的栅格单元。NearestNeighbor(最近邻), Bilinear, Cubic, CubicSpline 和 Lanczos 重新采样算法。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。Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed if no band is specified.
- **ST_Transform** - 2.0.0 新增函数。GDAL 1.6.1 版本。将栅格从一个坐标系转换到另一个坐标系。NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos 重新采样算法。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 (包括 NODATA)。
- **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 函数。将点添加到拓扑中 (split) 并返回面标识符。
- **TopoGeo_AddPolygon** - 2.0.0 函数。Adds a polygon to an existing topology using a tolerance and possibly splitting existing edges/faces. Returns face identifiers.
- **TopologySummary** - 2.0.0 函数。Takes a topology name and provides summary totals of types of objects in topology.
- **toTopoGeom** - 2.0 函数。Converts a simple Geometry into a topo geometry.
- **~** - 2.0.0 函数。A 与 B 是否相等 TRUE 或 FALSE。
- **~=** - 2.0.0 函数。A 与 B 是否相等 TRUE 或 FALSE。

Functions enhanced in PostGIS 2.0

- **&&** - 函数：2.0.0 函数 (polyhedral surface) 函数。A 与 B 是否相等 TRUE 或 FALSE。
- **AddGeometryColumn** - 函数：2.0.0 函数。use typmod 函数。函数 typmod 函数。函数。
- **Box2D** - 函数：2.0.0 函数, 函数 TIN 函数。Returns a BOX2D representing the 2D extent of a geometry.
- **Box3D** - 函数：2.0.0 函数, 函数 TIN 函数。Returns a BOX3D representing the 3D extent of a geometry.
- **CreateTopology** - Enhanced: 2.0 added support for hasZ. Creates a new topology schema and registers it in the topology.topology table.
- **GeometryType** - 函数：2.0.0 函数, 函数 TIN 函数。ST_Geometry 函数。
- **Populate_Geometry_Columns** - 函数：2.0.0 函数。函数, 函数 函数 use typmod 函数。Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- **ST_3DExtent** - 函数：2.0.0 函数, 函数 TIN 函数。Aggregate function that returns the 3D bounding box of geometries.
- **ST_Affine** - 函数：2.0.0 函数, 函数 TIN 函数。Apply a 3D affine transformation to a geometry.
- **ST_Area** - 函数：2.0.0 函数 2 函数 (polyhedral surface) 函数。函数。
- **ST_AsBinary** - 函数：2.0.0 函数, 函数 TIN 函数。Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST_AsBinary** - 函数：2.0.0 函数。Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST_AsBinary** - 函数：2.0.0 函数。Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST_AsEWKB** - 函数：2.0.0 函数, 函数 TIN 函数。Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.

- **ST_AsEWKT** - 函数: 2.0.0 将几何体转换为 WKT(Well-Known Text) 格式并返回 SRID 值。返回类型: text。
- **ST_AsGML** - 函数: 2.0.0 将几何体转换为 GML 3.1.1 格式并返回 SRID 值。返回类型: text。
- **ST_AsKML** - 函数: 2.0.0 将几何体转换为 KML 2.0 格式并返回 SRID 值。返回类型: text。
- **ST_Azimuth** - 函数: 2.0.0 计算从一点到另一点的方位角。返回类型: float。
- **ST_Dimension** - 函数: 2.0.0 返回几何体的维度 (polyhedral surface) 并返回 TIN 格式。返回类型: integer。
- **ST_Dump** - 函数: 2.0.0 将几何体分解为点、线、面并返回。Returns a set of geometry_dump rows for the components of a geometry.
- **ST_DumpPoints** - 函数: 2.0.0 将几何体分解为点并返回。返回类型: geometry_dump。
- **ST_Expand** - 函数: 2.0.0 将几何体的边界框扩展指定的距离。Returns a bounding box expanded from another bounding box or a geometry.
- **ST_Extent** - 函数: 2.0.0 返回几何体的边界框。Aggregate function that returns the bounding box of geometries.
- **ST_Force2D** - 函数: 2.0.0 将几何体强制转换为 2D (polyhedral surface) 并返回。返回类型: geometry。
- **ST_Force3D** - 函数: 2.0.0 将几何体强制转换为 3D (polyhedral surface) 并返回。返回类型: geometry。
- **ST_Force3DZ** - 函数: 2.0.0 将几何体强制转换为 3D (polyhedral surface) 并返回。返回类型: geometry。
- **ST_ForceCollection** - 函数: 2.0.0 将几何体强制转换为集合 (polyhedral surface) 并返回。返回类型: geometry。
- **ST_ForceRHR** - 函数: 2.0.0 将几何体强制转换为右手坐标系 (polyhedral surface) 并返回。返回类型: geometry。
- **ST_GMLToSQL** - 函数: 2.0.0 将 GML 格式转换为 SQL 格式 (polyhedral surface) 并返回。返回类型: geometry。
- **ST_GMLToSQL** - 函数: 2.0.0 将 GML 格式转换为 SQL 格式并返回 SRID 值。返回类型: geometry。
- **ST_GeomFromEWKB** - 函数: 2.0.0 从 EWKB(Extended Well-Known Binary) 格式创建几何体并返回。返回类型: geometry。
- **ST_GeomFromEWKT** - 函数: 2.0.0 从 EWKT(Extended Well-Known Text) 格式创建几何体并返回。返回类型: geometry。
- **ST_GeomFromGML** - 函数: 2.0.0 从 GML 格式创建几何体并返回。返回类型: geometry。
- **ST_GeomFromGML** - 函数: 2.0.0 从 GML 格式创建几何体并返回 SRID 值。返回类型: geometry。
- **ST_GeometryN** - 函数: 2.0.0 返回几何体的第 N 个分量。返回类型: geometry。
- **ST_GeometryType** - 函数: 2.0.0 返回几何体的类型 (polyhedral surface) 并返回。返回类型: text。

- ## Functions changed in PostGIS 2.0

- **AddGeometryColumn** - 新增: 2.0.0 新增 geometry columns 函数。该函数用于在表中添加 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 WGS84 POINT 函数中是有效的。该函数在 ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326); 语句中是有效的。
- **AddGeometryColumn** - 新增: 2.0.0 新增。该函数用于在表中添加 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 WGS84 POINT 函数中是有效的。该函数在 ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326); 语句中是有效的。
- **AddGeometryColumn** - 新增: 2.0.0 新增。该函数用于在表中添加 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 WGS84 POINT 函数中是有效的。该函数在 ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326); 语句中是有效的。
- **Box3D** - 新增: 2.0.0 新增 BOX3D 函数。BOX2D 函数在 2.0.0 版本中是有效的。BOX3D 函数在 2.0.0 版本中是有效的。
- **DropGeometryColumn** - 新增: 2.0.0 新增。该函数用于在表中删除 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 ALTER TABLE 语句中是有效的。
- **DropGeometryTable** - 新增: 2.0.0 新增。该函数用于在数据库中删除 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 DROP TABLE 语句中是有效的。
- **Populate Geometry Columns** - 新增: 2.0.0 新增。该函数用于在表中添加 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 ALTER TABLE 语句中是有效的。该函数在 WGS84 POINT 函数中是有效的。该函数在 ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326); 语句中是有效的。
- **ST_3DExtent** - 新增: 2.0.0 新增。该函数用于在表中添加 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 ALTER TABLE 语句中是有效的。
- **ST_3DLength** - 新增: 2.0.0 新增。该函数用于在表中添加 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 ALTER TABLE 语句中是有效的。
- **ST_3DMakeBox** - 新增: 2.0.0 新增。该函数用于在表中添加 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 ALTER TABLE 语句中是有效的。
- **ST_3DPerimeter** - 新增: 2.0.0 新增。该函数用于在表中添加 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 ALTER TABLE 语句中是有效的。
- **ST_AsBinary** - 新增: 2.0.0 新增。该函数用于在表中添加 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 ALTER TABLE 语句中是有效的。
- **ST_AsGML** - 新增: 2.0.0 新增。该函数用于在表中添加 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 ALTER TABLE 语句中是有效的。
- **ST_AsGeoJSON** - 新增: 2.0.0 新增。该函数用于在表中添加 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 ALTER TABLE 语句中是有效的。
- **ST_AsSVG** - 新增: 2.0.0 新增。该函数用于在表中添加 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 ALTER TABLE 语句中是有效的。
- **ST_EndPoint** - 新增: 2.0.0 新增。该函数用于在表中添加 geometry 列。该函数在 PostgreSQL 中是有效的。该函数在 ALTER TABLE 语句中是有效的。

- **ST_GDALDrivers** - 函数: 2.0.6, 2.1.3 新增 - GUC 参数 `gdal_enabled_drivers` 控制返回的栅格格式列表。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 新增。PostGIS 2.0.0 新增, SQL/MM 兼容。Returns a geometry object from a WKT string. `ST_GeomFromText('GEOMETRYCOLLECTION EMPTY')` returns a null geometry. WKT strings starting with `SRID=` are supported.
- **ST_GeometryN** - 函数: 2.0.0 新增。Returns a geometry object from a geometry collection. `ST_GeometryN(...,1)` returns the first geometry. `ST_Geometry` returns the first geometry.
- **ST_IsEmpty** - 函数: PostGIS 2.0.0 新增。Returns a boolean value indicating if a geometry is empty. PostGIS 2.0.0 新增, SQL/MM 兼容。Returns a boolean value indicating if a geometry is empty.
- **ST_Length** - 函数: 2.0.0 新增。Returns the length of a geometry. `ST_Length` returns the length of a geometry. `ST_Length` returns the length of a geometry. `ST_Length` returns the length of a geometry. `ST_Length` returns the length of a geometry.
- **ST_LocateAlong** - 函数: 2.0.0 新增。Returns the point(s) on a geometry that match a measure value. `ST_LocateAlong` returns the point(s) on a geometry that match a measure value.
- **ST_LocateBetween** - 函数: 2.0.0 新增。Returns the portions of a geometry that match a measure range. `ST_LocateBetween` returns the portions of a geometry that match a measure range.
- **ST_ModEdgeSplit** - 函数: 2.0 新增。Returns a geometry object from a geometry and a point. `ST_ModEdgeSplit` returns a geometry object from a geometry and a point.
- **ST_NumGeometries** - 函数: 2.0.0 新增。Returns the number of geometries in a geometry collection. `ST_NumGeometries` returns the number of geometries in a geometry collection.
- **ST_NumInteriorRings** - 函数: 2.0.0 新增。Returns the number of interior rings in a polygon. `ST_NumInteriorRings` returns the number of interior rings in a polygon.
- **ST_PointN** - 函数: 2.0.0 新增。Returns a point from a geometry. `ST_PointN` returns a point from a geometry. `ST_PointN` returns a point from a geometry. `ST_PointN` returns a point from a geometry.
- **ST_ScaleX** - 函数: 2.0.0 新增。Returns a geometry object from a geometry and a scale factor. `ST_ScaleX` returns a geometry object from a geometry and a scale factor.
- **ST_ScaleY** - 函数: 2.0.0 新增。Returns a geometry object from a geometry and a scale factor. `ST_ScaleY` returns a geometry object from a geometry and a scale factor.
- **ST_SetScale** - 函数: 2.0.0 新增。Returns a geometry object from a geometry and a scale factor. `ST_SetScale` returns a geometry object from a geometry and a scale factor.
- **ST_StartPoint** - 函数: 2.0.0 新增。Returns the first point of a LineString, CircularLineString, or NURBSCurve. `ST_StartPoint` returns the first point of a LineString, CircularLineString, or NURBSCurve.

13.12.15 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 支持从 GML 几何构造器。LibXML2 1.6 支持从 GML 几何构造器构造 PostGIS 几何构造器。
- **ST_GeomFromKML** - Availability: 1.5, requires libxml2 2.6+ 支持从 KML 几何构造器构造 PostGIS 几何构造器。
- **ST_HausdorffDistance** - 1.5.0 支持 Hausdorff 距离。返回两个几何体之间的最短距离 (shortest) 距离。
- **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 支持计算几何体的长度。返回几何体的长度。
- **ST_LongestLine** - 1.5.0 支持计算两个几何体之间的最长线。返回两个几何体之间的最长线 (longest) 线。
- **ST_MakeEnvelope** - 1.5 支持从 SRID 和 SRS 构造几何体。返回从 SRID 和 SRS 构造的几何体。
- **ST_MaxDistance** - 1.5.0 支持计算两个几何体之间的最大距离。返回两个几何体之间的最大距离。
- **ST_ShortestLine** - 1.5.0 支持计算两个几何体之间的最短线。返回两个几何体之间的最短线。
- **~=** - 1.5.0 支持计算两个几何体之间的相等性。A 与 B 相等返回 TRUE。

13.12.16 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 支持从 geometry 列构造几何体。Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- **ST_Collect** - 1.4.0 支持从几何体集合构造几何体。返回 ST_MakeLine 构造的几何体集合。Creates a GeometryCollection or Multi* geometry from a set of geometries.
- **ST_ContainsProperly** - 1.4.0 支持从几何体构造几何体。Tests if every point of B lies in the interior of A
- **ST_GeoHash** - 1.4.0 支持从几何体构造 GeoHash。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 支持从几何体构造几何体。Returns the portions of a geometry that lie in an elevation (Z) range.
- **ST_MakeLine** - 1.4.0 支持从几何体构造几何体。返回 ST_MakeLine 构造的几何体。Returns a geometry constructed from a set of geometries.
- **ST_MinimumBoundingCircle** - 1.4.0 支持从几何体构造几何体。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.

13.12.17 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 格式的数据转换为文本格式。
 - **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 3.7.0alpha1

This version requires GEOS 3.10+, PostgreSQL 14-19beta1, Proj 6.1+, libgmp. To take advantage of all features postgis extension features, GEOS 3.15+ is needed. To take advantage of all postgis_sfcgal extension features SFCGAL 2.3+ is needed.

A.1.1 Breaking Changes

#5688, [topology] topology building functions now interpret tolerance 0 (old default) to really mean 0 and want -1 (new default) to mean "use topology precision" (Sandro Santilli)

#6024, Remove support for PostgreSQL 12 and 13 (Regina Obe)

#6040, ST_GeoHash round trip through ST_GeomFromGeoHash (Paul Ramsey)

#6053, [address_standardizer] Extension removed and moved to https://github.com/postgis/address_standardizer (Paul Ramsey)

#6052, [tiger_geocoder] Extension removed and moved to https://git.osgeo.org/postgis/postgis_tiger_geocoder (Regina Obe)

#6079, Remove support for GEOS < 3.10 (Sandro Santilli)

#1793, Remove ST_MemCollect aggregate (Darafei Praliaskouski)

GH-848, CG_AlphaShape now returns a MultiPolygon (Jean Felder)

A.1.2 New Features

#1124, **#2935**, **#3033**, **#4658**, **#4659**, [loader] Rework loader arguments: shp2pgsql can create UNLOGGED tables and choose the feature id column name, shp2pgsql --drop-table can emit DROP TABLE before prepare output, raster2pgsql/shp2pgsql expose loader actions as long options, both loaders accept long aliases for existing options, and

-if-not-exists makes creation actions idempotent (Darafei Praliaskouski)

#4208, Add single-geometry variants of ST_MaxDistance and ST_LongestLine (Darafei Praliaskouski)

[topology] FindVertexSegmentPairsBelowDistance function (Sandro Santilli)

ST_CoverageEdges, returns MultiLinestring of distinct shared edges in polygonal coverage (Paul Ramsey)

ST_MinimumSpanningTree, window function to calculate MST (Paul Ramsey)

#5993, [topology] Add max_edges parameter to TopoGeo_AddLinestring (Sandro Santilli)

#6001, support MultiLineString in ST_MakeLine (Paul Ramsey)

#2858, ST_MMin and ST_MMax (Paul Ramsey)

#5941, [raster] Support GDT_Float16 pixel type (Darafei Praliaskouski)

#5992, Optimize GiST index for repeated edge subdivision in topology (Darafei Praliaskouski)

#5702, Allow the compiler to detect the parallelism -flto=auto (Darafei Praliaskouski)

#4798, ST_AsGeoJSON warns about duplicate property keys (Darafei Praliaskouski)

#5950, Document POSTGIS_REGRESS_DB_OWNER for sandboxed regression roles (Darafei Praliaskouski)

#4332, Clarify the scope of several Function Reference categories (Darafei Praliaskouski)

#4222, [raster] ST_DumpAsPolygons honours PostgreSQL interrupts (Darafei Praliaskouski)

#5109, Document the meaning of topology.next_left_edge and topology.next_right_edge links (Darafei Praliaskouski)

#5889, [topology] Include representative locations in topology build errors (Darafei Praliaskouski)

#2614, Use GEOSPreparedDistance and caching to accelerate ST_DWithin (Paul Ramsey)

#6022, Document ST_AsMVTGeom tile-coordinate simplification (Darafei Praliaskouski)

GH-839, ST_Multi support for TIN and surfaces (Loïc Bartoletti)

#4398, Add ST_CatmullSmoothing smoothes but retains original vertices (Paul Ramsey)

#4560, ST_3DInterpolatePoint for M interpolation from XYZ inputs (Paul Ramsey)

#1291, ST_MakePolygon support for curved rings (Darafei Praliaskouski)

GT-270, Add NURBSCurve (Loïc Bartoletti)

#2863, Add ST_XSize, ST_YSize, ST_ZSize, and ST_MSize dimension helpers (Darafei Praliaskouski)

[fuzzers] Centralize OSS-Fuzz build/dependency logic in PostGIS scripts, prefer requested CXX over pg_config --cc for internal C++ checks, and make fuzzer linking/runtime dependency packaging portable (Darafei Praliaskouski)

A.1.3 Enhancements

#2045, pgsql2shp query dumps no longer require temporary table privileges (Darafei Praliaskouski)

#1577, [loader] Report unsupported .zip archive input before trying to open shapefile sidecar files (Darafei Praliaskouski)

#3158, Move geometry and geography typmod bit helpers out of the public liblwgeom header (Darafei Praliaskouski)

#4678, [raster] Move st_sum4ma neighborhood callback to C (Darafei Praliaskouski)

#2116, [raster] Add ST_Value nearest-neighbor boundary options (Darafei Praliaskouski)

#2804, #4315, [raster] Support two-argument ST_MapAlgebra callbacks and pass callback call data as actual arguments (Darafei Praliaskouski)

#2898, Document SQL function cost tiers for contributors

#6062, [topology] Stop using recursive snapping, for improved robustness (Sandro Santilli)

#6065, Improved winding order computation robustness for rings having collapsed elements (Sandro Santilli)

- #3743, #4385, [raster loader] Document raster2pgsql -s FROM_SRID:SRID reprojection support and clarify index creation timing for append loads (Darafei Praliaskouski)
 - #2137, [raster] Reuse matching source raster overviews in raster2pgsql -l (Darafei Praliaskouski)
 - #4749, Use point-in-polygon predicate fast paths for point-only GeometryCollections (Darafei Praliaskouski)
 - #5532, Validate the manual against DocBook XMLSchema in check-xml (Darafei Praliaskouski)
 - #6087, Add a repository-owned CI status checker and static dashboard generator (Darafei Praliaskouski)
 - #2807, [raster] Preserve missing NODATA values in ST_MapAlgebra outputs (Darafei Praliaskouski)
 - #2832, [raster] Avoid padding single-tile raster2pgsql overviews when padding is not requested (Darafei Praliaskouski)
 - #2623, Generate a pgsql2shp GID field when exporting geometry-only shapefiles (Darafei Praliaskouski)
 - #2850, Preserve pgsql2shp numeric precision and scale in DBF metadata (Darafei Praliaskouski)
 - #2386, Add raster2pgsql man page (Darafei Praliaskouski)
- Build PostgreSQL extension modules with ``-fno-semantic-interposition`` when available so LTO can optimize same-DSO calls to exported PostGIS entry points directly instead of routing through the module PLT; in a local PostgreSQL 18 ``pgbench`` microbenchmark over 100k four-point groups this reduced ``ST_Collect`` aggregate latency by about 4% (Darafei Praliaskouski)
- #5973, Install PostGIS utilities under ``pg_config --bindir`` by default and document ``pg_config``-derived install paths more accurately (Darafei Praliaskouski)
 - #2808, Clarify `postgis_full_version()` output when installed component scripts are newer than core scripts (Darafei Praliaskouski)
 - #5593, Docs: render manual images with GraphicsMagick without temporary files, enabling parallel builds, keeping ImageMagick fallbacks working, and documenting the generator workflow for contributors (Darafei Praliaskouski)
 - #5645, Docs: keep code operators (`"=>"`) intact in translated manuals by enforcing verbatim CSS (Darafei Praliaskouski)
 - #1408, Docs: flatten transparent manual images onto white backgrounds for PDF builds while keeping HTML images transparent (Darafei Praliaskouski)

A.1.4 Bug Fixes

- #4616, Detect pgsql2shp query keywords case-insensitively (Darafei Praliaskouski)
 - #2838, Emit valid X3D Coordinate nodes for 2D polygon output (Darafei Praliaskouski)
- Fix WKB and TWKB parser resource exhaustion on malformed input (Darafei Praliaskouski)
- #6083, Pass configured dependency include paths to fuzzer smoke builds (Darafei Praliaskouski)
 - #3103, Add regression coverage for exact-schema `find_srid` and `geometry_columns` lookups (Darafei Praliaskouski)
 - #1705, Infer constraint metadata for direct view and materialized view geometry columns (Darafei Praliaskouski)
 - #6038, Avoid stale relation lookups in `geometry_columns` after topology objects are dropped (Darafei Praliaskouski)
 - #5655, [doc] Skip XML validation during ``make check`` when `xsftproc` is unavailable (Darafei Praliaskouski)
-

- #5487**, Improve error detail for repeated upgrades after an incomplete deprecated-function cleanup (Darafei Praliaskouski)
- GH-899**, [raster] Honor PostgreSQL interrupts in long-running GDAL progress callbacks (Darafei Praliaskouski)
- #5975**, Initialize skipped union-find cluster ids deterministically (Darafei Praliaskouski)
- #5904**, [topology] Avoid MinGW format warnings when logging topology ids (Darafei Praliaskouski)
- GH-885**, [topology] Avoid GMP overflow in ring orientation for very large finite coordinates (Darafei Praliaskouski)
- #6073**, **#6074**, Stabilize geography LRS endpoint interpolation and pole centroid regressions on 32-bit systems (Darafei Praliaskouski)
- #6082**, Fix Japanese PDF build by avoiding non-ASCII arrows in SQL program listings (Darafei Praliaskouski)
- GH-888**, [sfcgal] Avoid stale detoasted geometry access in CG_Visibility empty-input handling for SFCGAL < 2.2 (Darafei Praliaskouski)
- #3179**, Mark truncated libpgcommon PostgreSQL messages instead of silently dropping the tail (Darafei Praliaskouski)
- #6003**, Avoid size_t formats in liblwgeom logging/error wrappers for MinGW builds (Darafei Praliaskouski)
- GH-886**, ST_ClusterRelateWin validates NULL relate matrices before detoasting them (Darafei Praliaskouski)
- GH-889**, ST_Relate pattern inversion avoids out-of-bounds access on short DE-9IM patterns (Darafei Praliaskouski)
- GH-890**, Avoid GIDX BRIN summary merges across mismatched dimensions (Darafei Praliaskouski)
- #5948**, [topology] Prevent MakeTopologyPrecise from erasing edges when grid size exceeds their extent (Darafei Praliaskouski)
- #5959**, Prevent histogram target overflow when analysing massive tables (Darafei Praliaskouski)
- #6080**, Avoid out-of-bounds hex WKB lookup for high-bit input bytes (Darafei Praliaskouski)
- GH-895**, Qualify pg_class lookup during SRID/PROJ validation (Darafei Praliaskouski)
- #4828**, geometry_columns handles NOT VALID SRID checks without errors (Darafei Praliaskouski)
- #6048**, [raster] ST_Clip no longer crashes when clipping sparse band selections (Darafei Praliaskouski)
- GH-894**, [raster] Fix Float16 import, mixed nodata warping, band replacement cleanup, GDAL credential redaction, and Python GDAL compatibility (Darafei Praliaskouski)
- Flatgeobuf schema mismatch vulnerability (NeuroWinter)
- #5899**, pg_upgrade issue for non-standard geography SRID (Paul Ramsey)
- #2583**, Preserve Z/ZM dimensional qualifiers in ST_AsEWKT output for TIN (Darafei Praliaskouski)
-