

PostGIS 3.6.1 ☒☒☒☒☒☒

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

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

4 Data Management 29

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

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

7	PostGIS Reference	70
7.1	PostgreSQL PostGIS Geometry/Geography/Box 2D	70
7.1.1	box2d	70
7.1.2	box3d	71
7.1.3	geometry	71
7.1.4	geometry_dump	72
7.1.5	geography	72
7.2	管理几何列和表	73
7.2.1	AddGeometryColumn	73
7.2.2	DropGeometryColumn	75
7.2.3	DropGeometryTable	75
7.2.4	Find_SRID	76
7.2.5	Populate_Geometry_Columns	77
7.2.6	UpdateGeometrySRID	78
7.3	ST_ 函数 (constructor)	79
7.3.1	ST_Collect	79
7.3.2	ST_LineFromMultiPoint	81
7.3.3	ST_MakeEnvelope	82
7.3.4	ST_MakeLine	82
7.3.5	ST_MakePoint	84
7.3.6	ST_MakePointM	85
7.3.7	ST_MakePolygon	87
7.3.8	ST_Point	88
7.3.9	ST_PointZ	90
7.3.10	ST_PointM	90
7.3.11	ST_PointZM	91
7.3.12	ST_Polygon	91
7.3.13	ST_TileEnvelope	92
7.3.14	ST_HexagonGrid	93
7.3.15	ST_Hexagon	96
7.3.16	ST_SquareGrid	97
7.3.17	ST_Square	98
7.3.18	ST_Letters	99
7.4	几何类型访问器 (accessor)	100
7.4.1	GeometryType	100
7.4.2	ST_Boundary	102
7.4.3	ST_BoundingDiagonal	104
7.4.4	ST_CoordDim	105
7.4.5	ST_Dimension	105

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

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

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

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

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

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

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

7.14.2ST_SimplifyVW	429
7.14.26T_SetEffectiveArea	430
7.14.28T_TriangulatePolygon	432
7.14.28T_VoronoiLines	434
7.14.29T_VoronoiPolygons	435
7.15Coverages	437
7.15.1ST_CoverageInvalidEdges	437
7.15.2ST_CoverageSimplify	438
7.15.3ST_CoverageUnion	440
7.15.4ST_CoverageClean	441
7.16Affine Transformations	442
7.16.1ST_Affine	442
7.16.2ST_Rotate	444
7.16.3ST_RotateX	445
7.16.4ST_RotateY	446
7.16.5ST_RotateZ	447
7.16.6ST_Scale	448
7.16.7ST_Translate	449
7.16.8ST_TransScale	451
7.17Clustering Functions	452
7.17.1ST_ClusterDBSCAN	452
7.17.2ST_ClusterIntersecting	454
7.17.3ST_ClusterIntersectingWin	454
7.17.4ST_ClusterKMeans	455
7.17.5ST_ClusterWithin	457
7.17.6ST_ClusterWithinWin	458
7.18Bounding Box Functions	459
7.18.1Box2D	459
7.18.2Box3D	460
7.18.3ST_EstimatedExtent	461
7.18.4ST_Expand	462
7.18.5ST_Extent	463
7.18.6ST_3DExtent	464
7.18.7ST_MakeBox2D	466
7.18.8ST_3DMakeBox	466
7.18.9ST_XMax	467
7.18.10T_XMin	468
7.18.11T_YMax	469
7.18.12T_YMin	470

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

7.22.4	postgis.gdal_vsi_options	500
7.22.5	postgis.gdal_cpl_debug	501
7.23	Troubleshooting Functions	501
7.23.1	PostGIS_AddBBox	501
7.23.2	PostGIS_DropBBox	502
7.23.3	PostGIS_HasBBox	502
8	SFCGAL Functions Reference	504
8.1	SFCGAL Management Functions	504
8.1.1	postgis_sfcgal_version	504
8.1.2	postgis_sfcgal_full_version	504
8.2	SFCGAL Accessors and Setters	505
8.2.1	CG_ForceLHR	505
8.2.2	CG_IsPlanar	505
8.2.3	CG_IsSolid	506
8.2.4	CG_MakeSolid	506
8.2.5	CG_Orientation	507
8.2.6	CG_Area	507
8.2.7	CG_3DArea	508
8.2.8	CG_Volume	508
8.2.9	ST_ForceLHR	509
8.2.10	ST_IsPlanar	510
8.2.11	ST_IsSolid	510
8.2.12	ST_MakeSolid	511
8.2.13	ST_Orientation	511
8.2.14	ST_3DArea	512
8.2.15	ST_Volume	513
8.3	SFCGAL Processing and Relationship Functions	514
8.3.1	CG_Intersection	514
8.3.2	CG_Intersects	515
8.3.3	CG_3DIntersects	515
8.3.4	CG_Difference	516
8.3.5	ST_3DDifference	517
8.3.6	CG_3DDifference	518
8.3.7	CG_Distance	519
8.3.8	CG_3DDistance	520
8.3.9	ST_3DConvexHull	521
8.3.10	CG_3DConvexHull	521
8.3.11	ST_3DIntersection	522

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

9	拓扑 (topology)	572
9.1	拓扑操作	572
9.1.1	getfaceedges_returntype	572
9.1.2	TopoGeometry	573
9.1.3	validatetopology_returntype	573
9.2	拓扑元素	574
9.2.1	TopoElement	574
9.2.2	TopoElementArray	574
9.3	拓扑 TopoGeometry 操作	575
9.3.1	AddTopoGeometryColumn	575
9.3.2	RenameTopoGeometryColumn	576
9.3.3	DropTopology	577
9.3.4	RenameTopology	577
9.3.5	DropTopoGeometryColumn	578
9.3.6	FixCorruptTopoGeometryColumn	578
9.3.7	Populate_Topology_Layer	579
9.3.8	TopologySummary	580
9.3.9	ValidateTopology	581
9.3.10	ValidateTopologyRelation	583
9.3.11	ValidateTopologyPrecision	584
9.3.12	MakeTopologyPrecise	584
9.3.13	FindTopology	585
9.3.14	FindLayer	586
9.3.15	TotalTopologySize	586
9.3.16	UpgradeTopology	587
9.4	Topology Statistics Management	588
9.5	拓扑管理	588
9.5.1	CreateTopology	588
9.5.2	CopyTopology	589
9.5.3	ST_InitTopoGeo	590
9.5.4	ST_CreateTopoGeo	590
9.5.5	TopoGeo_AddPoint	591
9.5.6	TopoGeo_AddLineString	592
9.5.7	TopoGeo_AddPolygon	592
9.5.8	TopoGeo_LoadGeometry	593
9.6	拓扑函数	593
9.6.1	ST_AddIsoNode	593
9.6.2	ST_AddIsoEdge	594
9.6.3	ST_AddEdgeNewFaces	595

9.6.4	ST_AddEdgeModFace	595
9.6.5	ST_RemEdgeNewFace	596
9.6.6	ST_RemEdgeModFace	597
9.6.7	ST_ChangeEdgeGeom	597
9.6.8	ST_ModEdgeSplit	598
9.6.9	ST_ModEdgeHeal	599
9.6.10	ST_NewEdgeHeal	599
9.6.11	ST_MoveIsoNode	600
9.6.12	ST_NewEdgesSplit	601
9.6.13	ST_RemoveIsoNode	602
9.6.14	ST_RemoveIsoEdge	602
9.7	索引	603
9.7.1	GetEdgeByPoint	603
9.7.2	GetFaceByPoint	604
9.7.3	GetFaceContainingPoint	605
9.7.4	GetNodeByPoint	605
9.7.5	GetTopologyID	606
9.7.6	GetTopologySRID	606
9.7.7	GetTopologyName	607
9.7.8	ST_GetFaceEdges	608
9.7.9	ST_GetFaceGeometry	608
9.7.10	GetRingEdges	609
9.7.11	GetNodeEdges	610
9.8	索引	610
9.8.1	Polygonize	610
9.8.2	AddNode	611
9.8.3	AddEdge	612
9.8.4	AddFace	613
9.8.5	ST_Simplify	614
9.8.6	RemoveUnusedPrimitives	615
9.9	TopoGeometry 索引	615
9.9.1	CreateTopoGeom	615
9.9.2	toTopoGeom	617
9.9.3	TopoElementArray_Agg	619
9.9.4	TopoElement	619
9.10	TopoGeometry 索引	620
9.10.1	clearTopoGeom	620
9.10.2	TopoGeom_addElement	620
9.10.3	TopoGeom_remElement	621

9.10.4	TopoGeom_addTopoGeom	621
9.10.5	toTopoGeom	622
9.11	TopoGeometry 函数	622
9.11.1	GetTopoGeomElementArray	622
9.11.2	GetTopoGeomElements	623
9.11.3	ST_SRID	623
9.12	TopoGeometry 函数	624
9.12.1	AsGML	624
9.12.2	AsTopoJSON	627
9.13	拓扑函数	628
9.13.1	Equals	628
9.13.2	Intersects	629
9.14	Importing and exporting Topologies	629
9.14.1	Using the Topology exporter	630
9.14.2	Using the Topology importer	630
10	PostGIS 与云存储, 云存储	631
10.1	PostGIS 云存储	631
10.1.1	raster2pgsql 函数	631
10.1.1.1	Example Usage	631
10.1.1.2	raster2pgsql options	632
10.1.2	PostGIS 云存储	633
10.1.3	Using "out db" cloud rasters	634
10.2	PostGIS 云存储	635
10.2.1	PostGIS 云存储	635
10.2.2	PostGIS 云存储	636
10.3	PostGIS 云存储	637
10.3.1	PostGIS 云存储 ST_AsPNG 函数 PHP 函数	637
10.3.2	PostGIS 云存储 ST_AsPNG 函数 ASP.NET C# 函数	638
10.3.3	PostGIS 云存储 ST_AsPNG 函数 Java 函数	639
10.3.4	PLPython 函数 SQL 函数	641
10.3.5	PSQL 函数	641
11	PostGIS 云存储	643
11.1	PostGIS 云存储	644
11.1.1	geomval	644
11.1.2	addbandarg	644
11.1.3	rastbandarg	644
11.1.4	raster	645

11.1.5reclassarg	645
11.1.6summarystats	646
11.1.7unionarg	646
11.2索引 (constructor)	647
11.2.1AddRasterConstraints	647
11.2.2DropRasterConstraints	649
11.2.3AddOverviewConstraints	650
11.2.4DropOverviewConstraints	651
11.2.5PostGIS_GDAL_Version	651
11.2.6PostGIS_Raster_Lib_Build_Date	652
11.2.7PostGIS_Raster_Lib_Version	652
11.2.8ST_GDALDrivers	653
11.2.9UpdateRasterSRID	658
11.2.10ST_CreateOverview	658
11.3索引 (accessor)	659
11.3.1ST_AddBand	659
11.3.2ST_AsRaster	661
11.3.3ST_AsRasterAgg	663
11.3.4ST_Band	664
11.3.5ST_MakeEmptyCoverage	666
11.3.6ST_MakeEmptyRaster	667
11.3.7ST_Tile	668
11.3.8ST_Retile	671
11.3.9ST_FromGDALRaster	671
11.4索引 (accessor)	672
11.4.1ST_GeoReference	672
11.4.2ST_Height	673
11.4.3ST_IsEmpty	674
11.4.4ST_MemSize	674
11.4.5ST_MetaData	675
11.4.6ST_NumBands	676
11.4.7ST_PixelHeight	676
11.4.8ST_PixelWidth	677
11.4.9ST_ScaleX	679
11.4.10ST_ScaleY	679
11.4.11ST_RasterToWorldCoord	680
11.4.12ST_RasterToWorldCoordX	681
11.4.13ST_RasterToWorldCoordY	682
11.4.14ST_Rotation	683

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

11.7.3	ST_SetScale	726
11.7.4	ST_SetSkew	727
11.7.5	ST_SetSRID	728
11.7.6	ST_SetUpperLeft	728
11.7.7	ST_Resample	729
11.7.8	ST_Rescale	730
11.7.9	ST_Reskew	732
11.7.10	ST_SnapToGrid	733
11.7.11	ST_Resize	734
11.7.12	ST_Transform	735
11.8	ST_Band	738
11.8.1	ST_SetBandNoDataValue	738
11.8.2	ST_SetBandIsNoData	739
11.8.3	ST_SetBandPath	741
11.8.4	ST_SetBandIndex	742
11.9	ST_Stats	744
11.9.1	ST_Count	744
11.9.2	ST_CountAgg	744
11.9.3	ST_Histogram	746
11.9.4	ST_Quantile	747
11.9.5	ST_SummaryStats	749
11.9.6	ST_SummaryStatsAgg	751
11.9.7	ST_ValueCount	753
11.10	Raster Inputs	755
11.10.1	ST_RastFromWKB	755
11.10.2	ST_RastFromHexWKB	756
11.11	ST_As*	757
11.11.1	ST_AsBinary/ST_AsWKB	757
11.11.2	ST_AsHexWKB	757
11.11.3	ST_AsGDALRaster	758
11.11.4	ST_AsJPEG	759
11.11.5	ST_AsPNG	760
11.11.6	ST_AsTIFF	761
11.12	ST_MapAlgebra	762
11.12.1	ST_Clip	762
11.12.2	ST_ColorMap	766
11.12.3	ST_Grayscale	769
11.12.4	ST_Intersection	771
11.12.5	ST_MapAlgebra (callback function version)	773

11.12.6T_MapAlgebra (expression version)	779
11.12.3T_MapAlgebraExpr	782
11.12.8T_MapAlgebraExpr	784
11.12.9T_MapAlgebraFct	789
11.12.5T_MapAlgebraFct	793
11.12.5T_MapAlgebraFctNgb	797
11.12.5T_Reclass	799
11.12.5T_ReclassExact	801
11.12.5T_Union	803
11.13 索引	804
11.13.5T_Distinct4ma	804
11.13.8T_InvDistWeight4ma	805
11.13.5T_Max4ma	806
11.13.5T_Mean4ma	807
11.13.5T_Min4ma	808
11.13.6T_MinDist4ma	809
11.13.8T_Range4ma	810
11.13.8T_StdDev4ma	811
11.13.9T_Sum4ma	812
11.14 索引	813
11.14.5T_Aspect	813
11.14.8T_HillShade	815
11.14.8T_Roughness	817
11.14.5T_Slope	818
11.14.5T_TPI	819
11.14.6T_TRI	820
11.14.5T_InterpolateRaster	821
11.14.8T_Contour	822
11.15 索引	823
11.15.Box3D	823
11.15.8T_ConvexHull	823
11.15.5T_DumpAsPolygons	824
11.15.5T_Envelope	826
11.15.5T_MinConvexHull	826
11.15.6T_Polygon	827
11.15.8T_IntersectionFractions	829
11.16 索引	830
11.16.&&	830
11.16.&<	831

11.16.&>	831
11.16.4	832
11.16.5	833
11.16.6=	833
11.16.7	834
11.17 空间关系函数	834
11.17.\$T_Contains	834
11.17.\$T_ContainsProperly	835
11.17.\$T_Covers	836
11.17.\$T_CoveredBy	837
11.17.\$T_Disjoint	838
11.17.\$T_Intersects	839
11.17.\$T_Overlaps	840
11.17.\$T_Touches	841
11.17.\$T_SameAlignment	842
11.17.\$T_NotSameAlignmentReason	843
11.17.\$T_Within	844
11.17.\$T_DWithin	845
11.17.\$T_DFullyWithin	846
11.18 进阶提示	847
11.18.0 数据库中的栅格	847
11.18.1 包含许多文件的目录	847
11.18.1 最大打开文件数	847
11.18.1.2 整个系统的最大打开文件数	848
11.18.1.2 每个进程的最大打开文件数	848
12 PostGIS Extras	850
12.1 进阶函数	850
12.1.1 进阶函数	850
12.1.2 进阶函数	851
12.1.2.1 stdaddr	851
12.1.3 进阶函数	851
12.1.3.1 rules table	851
12.1.3.2 lex table	854
12.1.3.3 gaz table	855
12.1.4 进阶函数	855
12.1.4.1 debug_standardize_address	855
12.1.4.2 parse_address	857
12.1.4.3 standardize_address	858

12.2 TIGER 索引	859
12.2.1 Drop_Indexes_Generate_Script	860
12.2.2 Drop_Nation_Tables_Generate_Script	861
12.2.3 Drop_State_Tables_Generate_Script	861
12.2.4 Geocode	862
12.2.5 Geocode_Intersection	865
12.2.6 Get_Geocode_Setting	866
12.2.7 Get_Tract	867
12.2.8 Install_Missing_Indexes	868
12.2.9 Loader_Generate_Census_Script	868
12.2.10 Loader_Generate_Script	870
12.2.11 Loader_Generate_Nation_Script	872
12.2.12 Missing_Indexes_Generate_Script	873
12.2.13 Normalize_Address	874
12.2.14 Page_Normalize_Address	875
12.2.15 Print_Addy	877
12.2.16 Reverse_Geocode	878
12.2.17 Topology_Load_Tiger	880
12.2.18 Set_Geocode_Setting	882
13 PostGIS Special Functions Index	884
13.1 PostGIS Aggregate Functions	884
13.2 PostGIS Window Functions	885
13.3 PostGIS SQL-MM Compliant Functions	885
13.4 PostGIS Geography Support Functions	889
13.5 PostGIS Raster Support Functions	891
13.6 PostGIS Geometry / Geography / Raster Dump Functions	897
13.7 PostGIS Box Functions	897
13.8 PostGIS Functions that support 3D	898
13.9 PostGIS Curved Geometry Support Functions	904
13.10 PostGIS Polyhedral Surface Support Functions	907
13.1 PostGIS Function Support Matrix	911
13.11 New, Enhanced or changed PostGIS Functions	930
13.12.1 PostGIS Functions new or enhanced in 3.6	930
13.12.2 PostGIS Functions new or enhanced in 3.5	931
13.12.3 PostGIS Functions new or enhanced in 3.4	933
13.12.4 PostGIS Functions new or enhanced in 3.3	934
13.12.5 PostGIS Functions new or enhanced in 3.2	935
13.12.6 PostGIS Functions new or enhanced in 3.1	937

13.12.	PostGIS Functions new or enhanced in 3.0	938
13.12.	PostGIS Functions new or enhanced in 2.5	940
13.12.	PostGIS Functions new or enhanced in 2.4	941
13.12.	PostGIS Functions new or enhanced in 2.3	943
13.12.	PostGIS Functions new or enhanced in 2.2	945
13.12.	PostGIS Functions new or enhanced in 2.1	948
13.12.	PostGIS Functions new or enhanced in 2.0	954
13.12.	PostGIS Functions new or enhanced in 1.5	965
13.12.	PostGIS Functions new or enhanced in 1.4	967
13.12.	PostGIS Functions new or enhanced in 1.3	967
14	Reporting Problems	968
14.1	Reporting Software Bugs	968
14.2	Reporting Documentation Issues	968
A	Appendix	970
A.1	PostGIS 3.6.1	970
A.1.1	Fixes	970
A.2	PostGIS 3.6.0	970
A.2.1	Breaking Changes	971
A.2.2	Removed / Deprecate signatures	971
A.2.3	New Features	971

Abstract

PostGIS 是 PostgreSQL 数据库的 GIS(地理信息系统) 扩展。PostGIS 使用 GiST 和 R-Tree 索引，GIS 应用可以高效地处理空间数据。



PostGIS 3.6.1 版本发布。



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 应用可以高效地处理空间数据。
<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 项目成员

PostGIS 项目 Steering Committee (PSC) 是 PostGIS 项目的核心，负责项目的方向、决策、协调、沟通和文档。PSC 成员包括：PostGIS 项目的创始人、维护者和贡献者。PSC 成员的职责包括：API 设计、PostGIS 的测试、文档编写、项目协调、以及与其他 GIS 项目的合作。

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

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

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

Paul Ramsey (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 致谢 - 2019

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

Loïc Bartoletti SFCGAL enhancements and maintenance and ci support

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

Martin Davis GEOS enhancements and documentation

Björn Harrtell MapBox Vector Tile, GeoBuf, and Flatgeobuf functions. Gitea testing and GitLab experimentation.

Aliaksandr Kalenik Geometry Processing, PostgreSQL gist, general bug fixing

1.3 致谢 - 2020

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 致谢

	Alex Bodnar	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	Nikolai Berkoff
	Ayo Adesugba	Ingvild Nystuen	Nikita Shulga
	Barbara Phillipot	Jackie Leng	Norman Vine
	Ben Jubb	James Addison	Patricia Tozer
	Bernhard Reiter	James Marca	Rafal Magda
	Björn Esser	Jan Katins	Ralph Mason
	Brian Hamlin	Jan Tojnar	Rémi Cura
	Bruce Rindahl	Jason Smith	Richard Greenwood
	Bruno Wolff III	Jeff Adams	Robert Coup
	Bryce L. Nordgren	Jelte Fennema	Roger Crew
	Carl Anderson	Jim Jones	Ron Mayer
	Charlie Savage	Joe Conway	Sam Peters
安装与使用指南	Chris Mayo	Jonne Savolainen	Sebastiaan Couwenberg
	Christian Schroeder	Jose Carlos Martinez Llari	Sergei Shoulbakov
	Christoph Berg	Jörg Habenicht	Sergey Fedoseev
	Christoph Moench-Tegeder	Julien Rouhaud	Shinichi Sugiyama
	Dane Springmeyer	Kashif Rasul	Shoaib Burq
	Daniel Nylander	Klaus Foerster	Silvio Grosso
	Dapeng Wang	Kris Jurka	Stefan Corneliu Petrea
	Daryl Herzmann	Laurențiu Nicola	Steffen Macke
	Dave Fuhry	Laurenz Albe	Stepan Kuzmin
	安装与使用指南 (David Zwarg)	Lars Roessiger	Stephen Frost
	安装与使用指南 (David Zwarg)	Leo Hsu	Steven Ottens
	安装与使用指南 (David Zwarg)	Loic Dachary	Talha Rizwan
	Denys Kovshun	Luca S. Percich	Teramoto Ikuhiro
	Dian M Fay	Lucas C. Villa Real	Tom Glancy
	Dmitry Vasilyev	Maksim Korotkov	Tom van Tilburg
	Eduin Carrillo	Maria Arias de Reyna	Victor Collod
	Esteban Zimanyi	Marc Ducobu	Vincent Bre
	Eugene Antimirov	Mark Sondheim	Vincent Mora
	Even Rouault	Markus Schaber	Vincent Picavet
	Florian Weimer	Markus Wanner	Volf Tomáš
	Frank Warmerdam	Matt Amos	Zuo Chenwei
	George Silva	Matt Bretl	

PostGIS 安装与使用指南, 感谢以下社区成员。

- [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](#)
-

PostGIS ☒ ☒

2.1 ☐ ☐ ☐ ☐

PostGIS , PostGIS (Section 3.3)
 (Section 3.4) .

2.2 ☒☒☒☒☒☒☒☒☒☒☒☒

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

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

Note

GEOS 依赖于 PostgreSQL 依赖于 C++ 编译选项。



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

编译选项 C++ 编译选项。 编译选项 (编译选项) 依赖于 PostgreSQL 编译选项。

PostGIS 编译选项。 编译选项。

2.2.1 安装

PostGIS 编译选项 <https://download.osgeo.org/postgis/source/postgis-3.6.1.tar.gz> 编译选项。

```
wget https://download.osgeo.org/postgis/source/postgis-3.6.1.tar.gz
tar -xvzf postgis-3.6.1.tar.gz
cd postgis-3.6.1
```

编译选项 postgis-3.6.1 (编译选项) 编译选项。

编译选项, [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 postgis-3.6.1 编译选项。

```
./configure
```

2.2.2 编译选项

PostGIS 编译选项。

编译选项

- PostgreSQL 12 - 18. A complete installation of PostgreSQL (including server headers) is required. PostgreSQL is available from <https://www.postgresql.org> 18 .
For a full PostgreSQL / PostGIS support matrix and PostGIS/GEOS support matrix refer to <https://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS>
- GNU C 编译选项 (gcc). PostGIS 编译选项 ANSI C 编译选项 gcc 编译选项。
- GNU Make(gmake 编译选项 make). 编译选项 GNU make 编译选项 make 编译选项。 make -v 编译选项。 编译选项 make 编译选项 PostGIS Makefile 编译选项。
- Proj reprojection library. Proj 6.1 or above is required. The Proj library is used to provide coordinate reprojection support within PostGIS. Proj is available for download from <https://proj.org/> .
- GEOS geometry library, version 3.8.0 or greater, but GEOS 3.14+ is required to take full advantage of all the new functions and features. GEOS is available for download from <https://libgeos.org> .

- LibXML2, version 2.5.x or higher. LibXML2 is currently used in some imports functions (ST_GeomFromGML and ST_GeomFromKML). LibXML2 is available for download from <https://gitlab.gnome.org/GNOME/libxml2/-/releases>.
- JSON-C 0.9 . JSON-C  ST_GeomFromGeoJson  GeoJSON               . JSON-C  <https://github.com/json-c/json-c/releases/>                    .
- GDAL, version 3+ is preferred. This is required for raster support. <https://gdal.org/download.html>.
-                       .   PostgreSQL                     .                        <http://trac.osgeo.org/postgis/ticket/635>                       .



- Section 2.1 SFCGAL is required. shapefile shp2pgsql-gui GTK(GTK+2.0, 2.8+). <http://www.gtk.org/>.
- SFCGAL, 1.4.1 or higher is required and 2.1+ is needed to be able to use all functionality. SFCGAL can be used to provide additional 2D and 3D advanced analysis functions to PostgreSQL cf Chapter 8. And also allow to use SFCGAL rather than GEOS for some 2D functions provided by both backends (like ST_Intersection or ST_Area, for instance). A PostgreSQL configuration variable postgres.backend allow end user to control which backend he want to use if SFCGAL is installed (GEOS by default). Nota: SFCGAL 1.2 require at least CGAL 4.3 and Boost 1.54 (cf: <https://sfcgal.org>) <https://gitlab.com/sfcgal/SFCGAL/>.
- In order to build the Section 12.1 you will also need PCRE 1 or 2 <http://www.pcre.org> (which generally is already installed on nix systems). Section 12.1 will automatically be built if it detects a PCRE library, or you pass in a valid --with-pcre-dir=/path/to/pcre during configure.
- To enable ST_AsMVT protobuf-c library 1.1.0 or higher (for usage) and the protoc-c compiler (for building) are required. Also, pkg-config is required to verify the correct minimum version of protobuf-c. See [protobuf-c](#). By default, Postgis will use Wagyu 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-wagyu during the configure step.
- CUnit(CUnit). <http://cunit.sourceforge.net/>.
- DocBook(xsltproc) DocBook <http://www.docbook.org/>.
- DBLatex(dblatex) PDF DBLatex <http://dblatex.sourceforge.net/>.
- ImageMagick(convert) ImageMagick <http://www.imagemagick.org/>.

2.2.3

Makefile Makefile

./configure

PostGIS 9.5.0, PostgreSQL 9.5.0. **./configure** options, see the **configure** script.

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX. XXXXX --help XX --help=short XXXXXX  
XXXXXX.
```


--with-sfcgal=PATH 指定 PostGIS 的 sfcgal 库的路径。PATH 是 sfcgal-config 脚本的输出。

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

Note

PostGIS 的 SVN 仓库，包含最新的源代码。



./autogen.sh

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

运行 **tar** 命令，将 PostGIS 的源代码打包。运行 **configure** 脚本，生成 **./autogen.sh** 脚本。

2.2.4 编译

运行 Makefile 编译 PostGIS 的源代码。

make

运行 **make** 命令，编译 PostGIS 的源代码。编译成功后，显示 "PostGIS was built successfully. Ready to install."。

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

make comments

PostGIS 2.0 的源代码。包含 PostGIS 的源代码 (cheat sheet) html 文件。运行 **xsltproc** 命令，生成 doc 文件夹中的 topology_cheatsheet.html, tiger_geocoder_cheatsheet.html, raster_cheatsheet.html, postgis_cheatsheet.html 4 个 HTML 文件。

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

make cheatsheets

2.2.5 PostGIS Extensions 安装

PostgreSQL 9.1 的 PostGIS 扩展安装。

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

make comments

运行 **tar** 命令，将 PostGIS 的源代码打包。运行 **tar** 命令，将 PostGIS 的源代码打包。运行 **comments** 命令，生成 comments 文件。

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

```
cd extensions
cd postgis
make clean
```

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

**Note**

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

extension 安装 OS 安装 PostGIS 安装。 PostGIS binaries 安装。

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

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

Once you do that, you should see postgis, 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%' or name LIKE 'address%';
```

name	default_version	installed_version
address_standardizer	3.6.1	3.6.1
address_standardizer_data_us	3.6.1	3.6.1
postgis	3.6.1	3.6.1
postgis_raster	3.6.1	3.6.1
postgis_sfcgal	3.6.1	
postgis_tiger_geocoder	3.6.1	3.6.1
postgis_topology	3.6.1	

(6 rows)

extension 安装, installed_version 安装。 postgis extension 安装。 PgAdmin III 1.14 安装 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 fuzzystmatch; --needed for postgis_tiger_geocoder
--optional used by postgis_tiger_geocoder, or can be used standalone
CREATE EXTENSION address_standardizer;
```

```
CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION postgis_topology;
```

PSQL 安装完成后，在 PostgreSQL 中安装 PostGIS 扩展。

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

List of installed extensions

```
-[ RECORD 1 ]-----
Name          | postgis
Version       | 3.6.1
Schema        | public
Description   | PostGIS geometry, geography, and raster spat..
-[ RECORD 2 ]-----
Name          | postgis_raster
Version       | 3.0.0dev
Schema        | public
Description   | PostGIS raster types and functions
-[ RECORD 3 ]-----
Name          | postgis_tiger_geocoder
Version       | 3.6.1
Schema        | tiger
Description   | PostGIS tiger geocoder and reverse geocoder
-[ RECORD 4 ]-----
Name          | postgis_topology
Version       | 3.6.1
Schema        | topology
Description   | PostGIS topology spatial types and functions
```

Warning



PostGIS 3.6.1 安装时，会安装以下扩展：postgis, postgis_raster, postgis_topology, postgis_tiger_geocoder, postgis_topology。PostGIS 2.0.1 安装时，会安装以下扩展：postgis, postgis_raster, postgis_topology, postgis_tiger_geocoder, postgis_topology。PostGIS 2.0.1 安装时，会安装以下扩展：postgis, postgis_raster, postgis_topology, postgis_tiger_geocoder, postgis_topology。PostGIS 2.0.1 安装时，会安装以下扩展：postgis, postgis_raster, postgis_topology, postgis_tiger_geocoder, postgis_topology。

PostGIS 3.6.1 安装时，会安装以下扩展：postgis, postgis_raster, postgis_topology, postgis_tiger_geocoder, postgis_topology。PostGIS 2.0.1 安装时，会安装以下扩展：postgis, postgis_raster, postgis_topology, postgis_tiger_geocoder, postgis_topology。

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

2.2.6 验证

验证 PostGIS 安装是否成功，可以使用以下命令。

make check

在 PostgreSQL 中执行以下命令，验证 PostGIS 安装是否成功。

**Note**

PostgreSQL, GEOS, 以及 Proj4 安装路径, LD_LIBRARY_PATH 环境变量.

**Caution**

安装时, **make check** 需要设置 PATH 和 PGPORT 环境变量. PostgreSQL 安装时, 需要设置 **--with-pgconfig** 选项. 安装时, PostgreSQL 安装路径需要设置 PATH 环境变量.

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

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

.
.
.

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

.
.
.

Running tests

.
.
.

Run tests: 134
Failed: 0

-- if you build with SFCGAL

.
.
.

Running tests

.
.
.

Run tests: 13
Failed: 0

-- if you built with raster support

.
```



```

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

.
.
.
Running tests

.
.
.
Run tests: 101
Failed: 0

-- topology regress

.
.
.
Running tests

.
.
.
Run tests: 51
Failed: 0

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

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

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

```

postgis_tiger_geocoder 与 address_standardizer 均需要 PostgreSQL 10.0 及以上 (installcheck) 才能通过。安装时请确保 PostgreSQL 10.0 及以上版本。安装: 安装 PostGIS 3.6.1 安装地址: <http://trac.osgeo.org/postgis/wiki/UsersWishList>。

address_standardizer 安装:

```

cd extensions/address_standardizer
make install
make installcheck

```

安装完成:

```

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

=====
All 4 tests passed.
=====

```

TIGER 数据加载器, 安装 PostgreSQL 的 PostGIS 扩展 `fuzzystrmatch`、`address_standardizer` 和 `address_standardizer`。

```

cd extensions/postgis_tiger_geocoder
make install
make installcheck

```

安装和测试:

```

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

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

```

2.2.7 安装

PostGIS 安装和配置。

make install

安装 `--prefix` 选项指定 PostgreSQL 安装目录。

- `loader` (loader) 文件 [prefix]/bin 目录。
- `postgis.sql` SQL 脚本 [prefix]/share/contrib 目录。
- PostGIS 库文件 [prefix]/lib 目录。

运行 `postgis_comments.sql`, `raster_comments.sql` 和 `make comments` 脚本，安装完成后，运行 `sql` 脚本。

make comments-install



Note

xsltproc 是运行 `postgis_comments.sql`, `raster_comments.sql`, `topology_comments.sql` 脚本所必需的。

2.3 安装 Tiger Geocoder

`address_standardizer` 是 PostGIS 2.2 版本引入的。它依赖于 `libaddressstandardizer`。安装 `libaddressstandardizer` 的说明在 Section 2.1 中。

运行 **Normalize Address** 是 PostGIS 2.2 版本引入的 Tiger Geocoder (geocoder) 的扩展。它依赖于 `libaddressstandardizer`。Section 2.4.2 中介绍了 `libaddressstandardizer` 的构建过程 (building block)。

安装 `PCRE` 是必需的。PCRE 的官方网站是 <http://www.pcre.org>。Section 2.2.3 中介绍了 `PCRE` 的安装。运行 `--with-pcre=/path/to/pcre` 和 `/path/to/pcre` 是 `PCRE` 的 include 和 lib 目录。

运行 `PostGIS 2.1` 版本引入的 `address_standardizer` 扩展。运行 `CREATE EXTENSION` 脚本。

运行以下 SQL 脚本：

```
CREATE EXTENSION address_standardizer;
```

运行 `rules`, `gaz`, 和 `lex` 脚本。

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

运行以下 SQL 脚本：

```
num | street | city | state | zip
----+-----+-----+-----+-----
1 | Devonshire Place PH301 | Boston | MA | 02109
```

2.4 Installing, Upgrading Tiger Geocoder, and loading data

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

2.4.1 Tiger Geocoder Enabling your PostGIS database

1. These directions assume your PostgreSQL installation already has the `postgis_tiger_geocoder` extension installed.
2. PSQL, pgAdmin 或者 任何 SQL 客户端。 PostGIS 安装后，安装 `fuzzystrmatch` 和 `postgis_tiger_geocoder`。

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

安装 `postgis_tiger_geocoder` 扩展：

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

`tiger.loader_platform` 表 `tiger.loader_variables` 存储了加载器的配置。

3. 使用 SQL 查询：

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

输出结果：

address	streetname	streotypeabbrev	zip
1 Devonshire Pl			02109

4. `tiger.loader_platform` 表存储了加载器的配置。使用 `sh` (convention) 或者 `debbie` 表。

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

使用 `debbie` 表，pg, unzip, shp2pgsql, PSQL 表 `declare_sect` 表。

使用 `loader_platform` 表，(common case) 表。

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

To enable it, do the following:

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

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

6. 在 Windows 上, 将 tiger_loader_variables 文件复制到 PC 上的 gisdata 目录中。在 Linux 上, 将 TIGER 数据文件复制到 /usr/share/postgis 目录中。在 Mac 上, 将 tiger_loader_variables 文件复制到 staging_fold 目录中。

7. gisdata 目录中的 staging_fold 目录中的 temp 目录中的 tiger_loader_variables 文件复制到 temp 目录中。

8. Then run the **Loader_Generate_Nation_Script** SQL function make sure to use the name of your custom profile and copy the script to a .sh or .bat file. So for example to build the nation load:

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

9. Run the generated nation load commandline scripts.

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

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

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

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

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

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

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

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

```
count
-----
   33933
(1 row)
```

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

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

Alternatively you can load just these tables after loading state data using the **Loader_Generate_Census**

12. For each state you want to load data for, generate a state script **Loader_Generate_Script**.

- PostGIS 安装 shp2pgsql 工具
- 使用 `wget` 在 Unix/Linux 系统上下载。

安装 `wget` <http://gnuwin32.sourceforge.net/packages/wget.htm>。

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

安装 `Loader_Generate_Nation_Script` 脚本。该脚本将生成 nation 表的 loader 脚本。该脚本将生成 nation 表的 loader 脚本。该脚本将生成 nation 表的 loader 脚本。

安装 `Install_Missing_Indexes` 脚本：

```
SELECT install_missing_indexes();
```

安装 `Geocode` 脚本。

2.4.4 Upgrading your Tiger Geocoder Install and Data

First upgrade your `postgis_tiger_geocoder` extension as follows:

```
ALTER EXTENSION postgis_tiger_geocoder UPDATE;
```

安装 `nation` 表。使用 `drop` 命令。SQL 命令 `drop` 命令。使用 `Drop_Nation_Tables_Generate_Script` 脚本。

```
SELECT drop_nation_tables_generate_script();
```

使用 `drop` 命令 SQL 命令。

使用 `SELECT` 命令 `nation` 加载。使用 `Loader_Generate_Nation_Script` 脚本。

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

unix/linux

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

Refer to Section 2.4.1 for instructions on how to run the generate script. This only needs to be done once.



Note

You can have a mix of different year state tables and can upgrade each state separately. Before you upgrade a state you first need to drop the prior year state tables for that state using `Drop_State_Tables_Generate_Script`.

2.5 安装与升级

安装与升级。

1. PostgreSQL 12 安装与配置。安装 PostgreSQL 12 需要安装 PostgreSQL 12 的包。 (Linux) 安装 PostgreSQL 12 需要安装 PostgreSQL 12 的包, 安装 PostgreSQL 12 的包。 PostGIS 3.6.1 需要 PostgreSQL 12 的包。 PostgreSQL 12 的包。 PostgreSQL 12 的包。 psql 安装 PostgreSQL 12 的包。

```
SELECT version();
```

RPM 安装 rpm 包: **rpm -qa | grep postgresql**

2. 安装 PostGIS 3.6.1。

```
SELECT postgis_full_version();
```

PostgreSQL, Proj4 安装 GEOS 安装。

1. 安装 postgis_config.hh 安装。 POSTGIS_PGSQL_VERSION, POSTGIS_PROJ_VERSION and POSTGIS_GEOS_VERSION 安装。

Chapter 3

PostGIS Administration

3.1 Performance Tuning

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

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

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

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

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

3.1.1 Startup

These settings are configured in `postgresql.conf`:

`constraint_exclusion`

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

`shared_buffers`

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

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

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

3.1.2 Runtime

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

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

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

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

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

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

max_parallel_workers_per_gather

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

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

3.2 Configuring raster support

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

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

If you want to enable offline raster:

```
POSTGIS_ENABLE_OUTDB_RASTERS=1
```

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

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

```
POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

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

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



Note

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

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

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

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

3.3 安装与配置

3.3.1 Spatially enable database using EXTENSION

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

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

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

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

3.3.2 Spatially enable database without using EXTENSION (discouraged)



Note

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

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

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

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

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

Run the following Shell commands in your terminal:

```
DB=[yourdatabase]
SCRIPTSDIR=`pg_config --sharedir`/contrib/postgis-3.5/

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

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

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

3.4 Upgrading spatial databases

Upgrading existing spatial databases can be tricky as it requires replacement or introduction of new PostGIS object definitions.

Unfortunately not all definitions can be easily replaced in a live database, so sometimes your best bet is a dump/reload process.

PostGIS provides a SOFT UPGRADE procedure for minor or bugfix releases, and a HARD UPGRADE procedure for major releases.

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

3.4.1 Soft upgrade

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

3.4.1.1 Soft Upgrade 9.1+ using extensions

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

If you are running PostGIS 3 or above, then you should use the `PostGIS_Extensions_Upgrade` function to upgrade to the latest version you have installed.

```
SELECT postgis_extensions_upgrade();
```

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

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

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

```
ALTER EXTENSION postgis UPDATE TO "3.6.1";  
ALTER EXTENSION postgis_topology UPDATE TO "3.6.1";
```

If you get an error notice something like:

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

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

If you get a notice message like:

```
Version "3.6.1" of extension "postgis" is already installed
```

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

```
ALTER EXTENSION postgis UPDATE TO "3.6.1next";  
ALTER EXTENSION postgis_topology UPDATE TO "3.6.1next";
```

**Note**

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

**Note**

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

```
DROP EXTENSION postgis_raster;
```

3.4.1.2 Soft Upgrade Pre 9.1+ or without extensions

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

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

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

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

```
ls `pg_config --sharedir`/contrib/postgis-3.6.1/*_upgrade.sql
```

Load them all in turn, starting from postgis_upgrade.sql.

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

The same procedure applies to raster, topology and sfcgal extensions, with upgrade files named rtpostgis_upgrade.sql, topology_upgrade.sql and sfcgal_upgrade.sql respectively. If you need them:

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

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

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

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

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

**Note**

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

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

3.4.2 Hard upgrade

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

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

Supplementary instructions for windows users are available at **Windows Hard upgrade**.

The Procedure is as follows:

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

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

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

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

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

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

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

Errors may arise in the following cases:

1. Some of your views or functions make use of deprecated PostGIS objects. In order to fix this you may try loading legacy.sql script prior to restore or you'll have to restore to a version of PostGIS which still contains those objects and try a migration again after porting your code. If the legacy.sql way works for you, don't forget to fix your code to stop using deprecated functions and drop them loading uninstall_legacy.sql.
2. Some custom records of spatial_ref_sys in dump file have an invalid SRID value. Valid SRID values are bigger than 0 and smaller than 999000. Values in the 999000.999999 range are reserved for internal use while values > 999999 can't be used at all. All your custom records with invalid SRIDs will be retained, with those > 999999 moved into the reserved range, but the spatial_ref_sys table would lose a check constraint guarding for that invariant to hold and possibly also its primary key (when multiple invalid SRIDS get converted to the same reserved SRID value).

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

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

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

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

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

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

```
postgis_restore "/somepath/olddb.backup" > olddb.sql
```

run this command :

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

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

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

Chapter 4

Data Management

4.1 GIS (地理) 数据库

4.1.1 OGC Geometry

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

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

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

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

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

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

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

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

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

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

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

4.1.1.1 Point

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

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

4.1.1.2 LineString

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

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

4.1.1.3 LinearRing

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

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

4.1.1.4 Polygon

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

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

4.1.1.5 MultiPoint

A MultiPoint is a collection of Points.

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

4.1.1.6 MultiLineString

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

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

4.1.1.7 MultiPolygon

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

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

4.1.1.8 GeometryCollection

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

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

4.1.1.9 PolyhedralSurface

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

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

4.1.1.10 Triangle

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

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

4.1.1.11 TIN

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

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

4.1.2 SQL-MM Part 3

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



Note

SQL-MM 对浮点精度有严格要求。精度为 1E-8 或更高。

4.1.2.1 CircularString

CIRCULARSTRING 是 LINESTRING 的子类型。它由一个或多个圆弧组成，每个圆弧由三个点定义。每个点由 X、Y 和 Z 坐标组成。Z 坐标是可选的。CIRCULARSTRING 的语法如下：

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

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

4.1.2.2 CompoundCurve

COMPOUNDCURVE (复合曲线) 是由一个或多个 CIRCULARSTRING 组成的。它的语法如下：

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

4.1.2.3 CurvePolygon

CURVEPOLYGON 是由一个或多个 COMPOUNDCURVE 组成的。它的语法如下：

PostGIS 1.4 支持 CURVEPOLYGON。

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

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

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

4.1.2.4 MultiCurve

MULTICURVE 数据类型, 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 是两种不同的几何数据表示格式。

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

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

Input and output of WKT is provided by the functions **ST_AsText** and **ST_GeomFromText**:

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

OGC 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');
```

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: 01020000000200000000000000000000004000000000000000040000000000000022400000000000002

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

```
INSERT INTO geotable ( geom, name )
VALUES ( ST_GeomFromWKB('\x0101000000000000000000f03f000000000000f03f', 312), 'A Place');
```

4.2 Geometry Data Type

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

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



Note

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

OpenGIS (SRID)

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

4.2.1 OpenGIS WKB ☒ WKT

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

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 (以及 ISO 15926) 在 WKB/WKT 与 EWKB/EWKT 格式上存在差异。PostGIS 与 OGC 规范在几何类型命名上也存在差异。PostGIS 与 OGC 规范在几何类型命名上也存在差异。PostGIS 与 OGC 规范在几何类型命名上也存在差异。

PostGIS 与 OGC 规范在几何类型命名上也存在差异 (WKT) 与 OGC 规范在几何类型命名上也存在差异:

- POINT(0 0 0) -- XYZ
- SRID=32632;POINT(0 0) -- SRID 32632 XY
- POINTM(0 0 0) -- XYM
- POINT(0 0 0 0) -- XYZM
- SRID=4326;MULTIPOINTM(0 0 0,1 2 1) -- SRID 4326 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 3.6.1 安装与使用

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

PostGIS 3.6.1 安装与使用

```
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

PostGIS 3.6.1 安装与使用

PostGIS 3.6.1 安装与使用

PostGIS 3.6.1 安装与使用

PostGIS 3.6.1 安装与使用

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

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

4.3.2 PostGIS 地理数据类型

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

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

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

```
-- NAD 83 lon/lat
SELECT 'SRID=4269;POINT(-123 34)::geography;
        geography
-----
0101000020AD100000000000000000C05EC000000000000004140
```

```
-- NAD27 lon/lat
SELECT 'SRID=4267;POINT(-123 34)::geography;
        geography
-----
0101000020AB100000000000000000C05EC000000000000004140
```

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

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

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

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

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

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

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

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

```
-- Distance calculation using GEOMETRY
SELECT ST_Distance('LINESTRING(-122.33 47.606, 0.0 51.5)::geometry, 'POINT(-21.96 64.15) ←
        '::geometry);
        st_distance
-----
13.342271221453624
```

4.3.3 性能优化建议

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

1. 数据库引擎配置：确保数据库引擎（如 PostgreSQL）配置正确，特别是内存和缓存设置。2. 硬件配置：确保有足够的内存和快速的 I/O 系统。3. 索引：为经常查询的列创建索引。4. 分区：对于大型表，考虑分区以改善查询性能。

- 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。
- 硬件配置：确保有足够的内存和快速的 I/O 系统。
- 索引：为经常查询的列创建索引。
- 分区：对于大型表，考虑分区以改善查询性能。

有关性能优化的详细讨论，请参考 Section 13.11。有关索引的讨论，请参考 Section 13.4。

4.3.4 常见问题 FAQ

1. 为什么我的查询很慢？
可能的原因包括：缺少索引、查询语句复杂、数据量大、硬件配置不足等。请检查查询语句，确保必要的索引存在，并考虑优化查询逻辑。
2. 如何设置 PostGIS 的日志？
PostGIS 的日志配置通常位于数据库的配置文件（如 postgresql.conf）中。可以通过设置 log_level 和 log_destination 来控制日志的输出。
3. 什么是“大圆弧”（great circle arc）？
“大圆弧”是指地球表面上两点之间的最短路径。在 PostGIS 中，可以使用 ST_ShortestPath 函数来计算大圆弧距离。
4. 为什么我的查询返回了错误的结果？
可能的原因包括：数据类型不匹配、查询语句中的逻辑错误、数据不一致等。请仔细检查查询语句，并确保数据的一致性和完整性。

4.4 Geometry Validation

PostGIS 是符合开放地理空间联盟（OGC）简单要素规范（Simple Features specification）的。该规范定义了“简单”（simple）和“有效”（valid）的几何概念。这些定义允许我们验证几何数据的有效性。

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

4.4.1 Simple Geometry

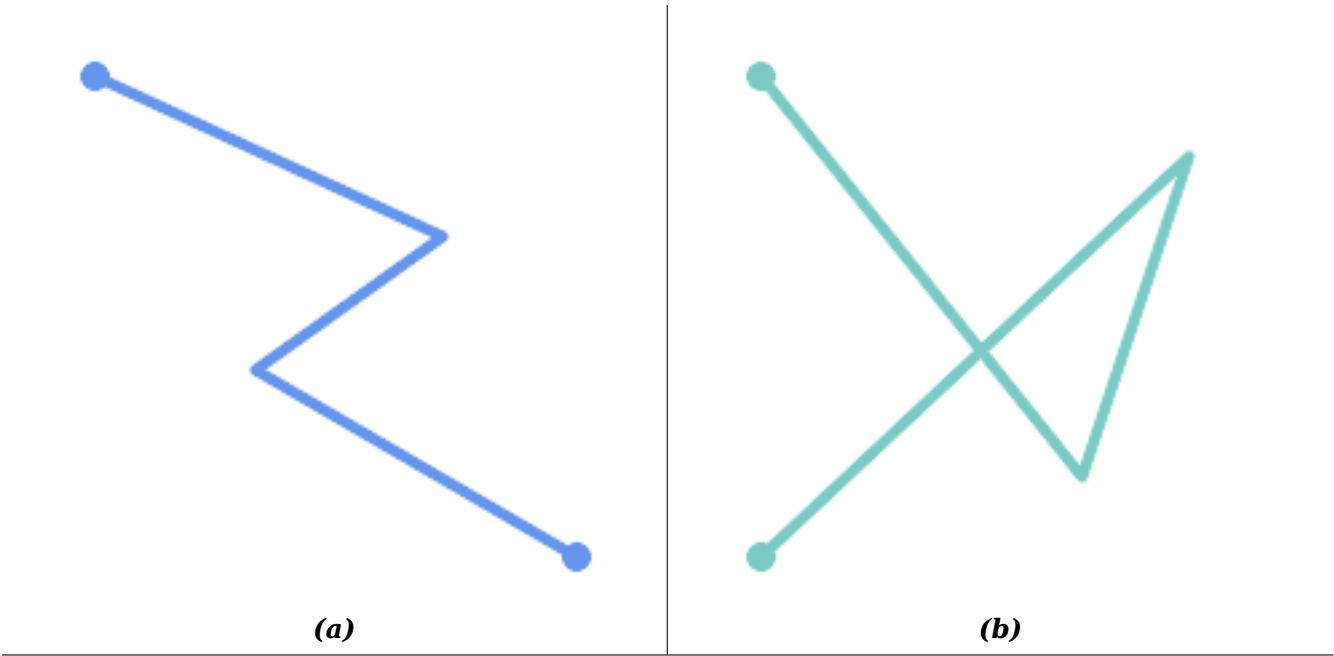
A *simple* geometry is one that has no anomalous geometric points, such as self intersection or self tangency.

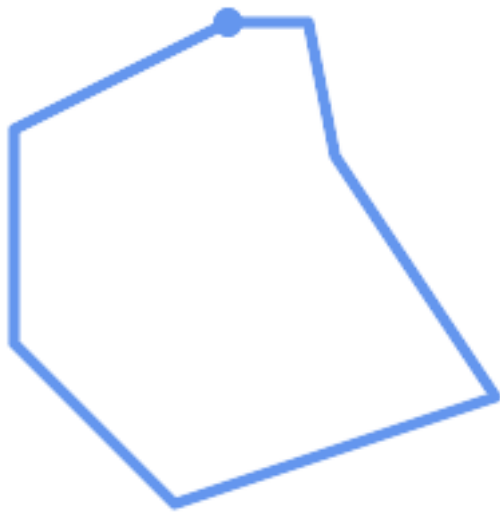
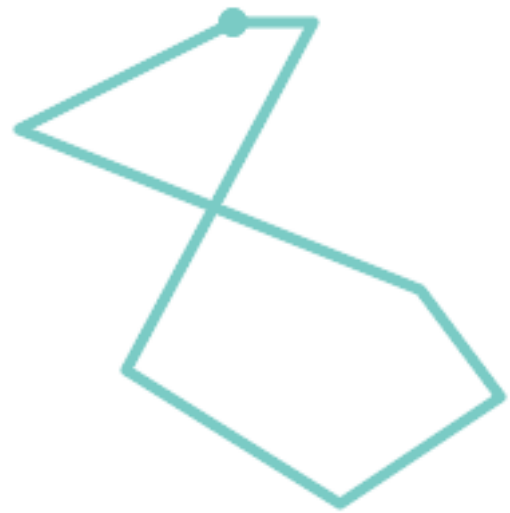
POINT 0 1000000000000000 1000000.

MULTIPOINT (POINT) 1000000000000000 (1000000000000000) 1000000.

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

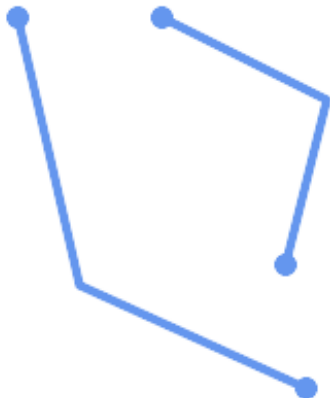
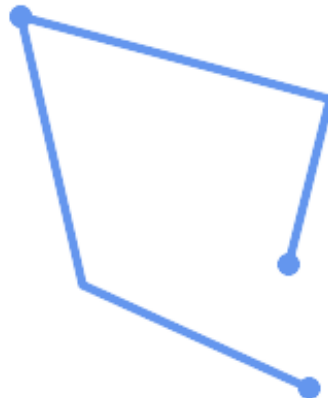
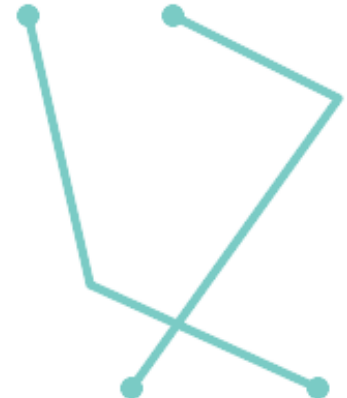
(a) and **(c)** are simple LINESTRINGs. **(b)** and **(d)** are not simple. **(c)** is a closed Linear Ring.



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

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

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

**(e)****(f)****(g)**

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

To test if a geometry is simple use the **ST_IsSimple** function:

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

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

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

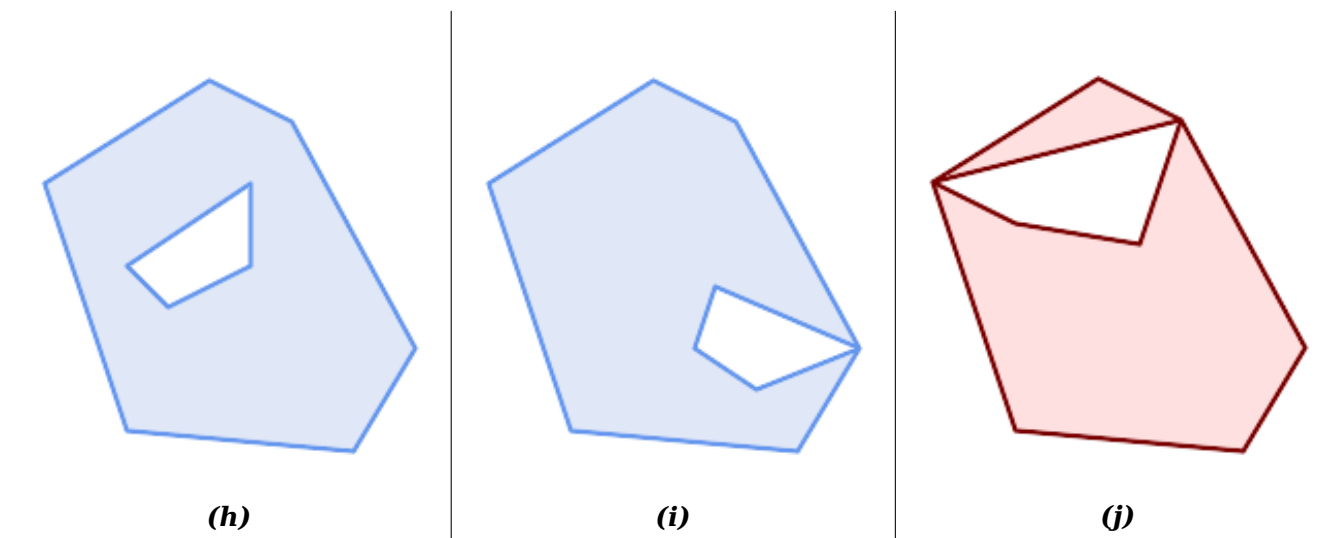
4.4.2 Valid Geometry

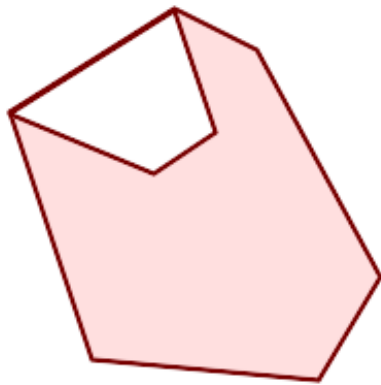
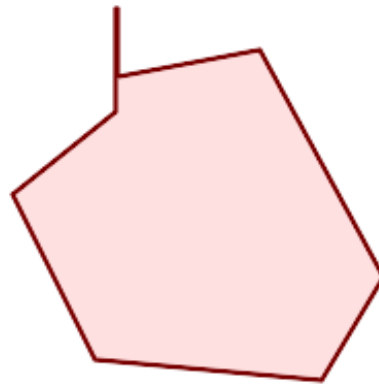
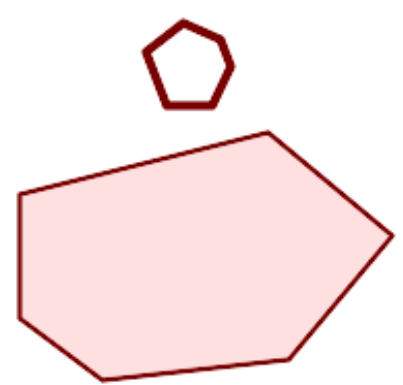
Geometry validity primarily applies to 2-dimensional geometries (POLYGONs and MULTIPOLYGONs). Validity is defined by rules that allow polygonal geometry to model planar areas unambiguously.

A POLYGON is *valid* if:

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

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

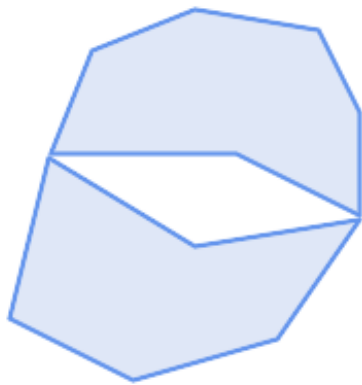
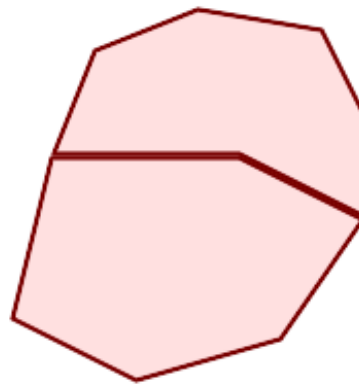
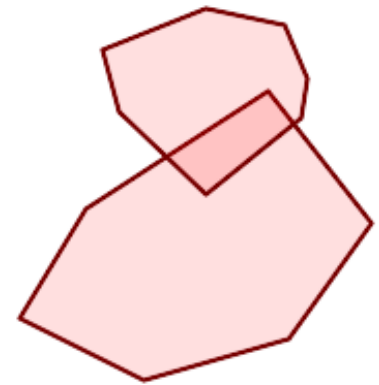


**(k)****(l)****(m)**

A MULTIPOLYGON is *valid* if:

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

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

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

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

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

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

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

POINT and MULTIPOINT geometries have no validity rules.

4.4.3 Managing Validity

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

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

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



Note

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

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

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

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

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

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

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

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

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

4.5 SPATIAL_REF_SYS 国际化

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 国际化:

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

国际化:

srid 国际化 (SRS) 国际化.

auth_name 国际化. 国际化“EPSG” 国际化.
AUTH_NAME 国际化.

auth_srid The ID of the Spatial Reference System as defined by the Authority cited in the auth_name. In the case of EPSG, this is the EPSG code.

srtext 国际化 WKT(Well-Known Text) 国际化. 国际化 WKT SRS 国际化:


```

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

```

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

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

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

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

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

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

4.5.2 SPATIAL_REF_SYS 国际化

PostGIS 的 SPATIAL_REF_SYS 存储 proj 字符串，存储 3000 个字符串，存储 proj4 字符串。 示例：

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

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

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

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

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

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

4.6 数据类型

4.6.1 几何数据类型

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

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

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

- **几何类型修饰符**. POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON. 修饰符 Z, M 或 ZM 指定 3D 几何体。例如，`geometry('LINESTRINGM', 3005)` 指定 3D 几何体。修饰符 `'POINTZM'` 指定 3D 几何体。
- the **SRID modifier** restricts the **spatial reference system** SRID to a particular number. If omitted, the SRID defaults to 0.

Examples of creating tables with geometry columns:

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

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

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

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

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

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

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

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

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

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

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

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

4.6.2 The GEOMETRY_COLUMNS VIEW

OpenGIS “SQL 特征规格 (Simple Features Specification for SQL)” 定义了 GIS 数据, 包括表、视图、索引、函数、操作符、数据类型、以及元数据。OpenGIS 规范定义了 OpenGIS 数据库系统必须实现的功能。

```
\d geometry_columns
```

```
View "public.geometry_columns"
  Column          |          Type          | Modifiers
-----+-----+-----
 f_table_catalog | character varying(256) |
 f_table_schema  | character varying(256) |
 f_table_name    | character varying(256) |
 f_geometry_column | character varying(256) |
 coord_dimension | integer                 |
 srid            | integer                 |
 type            | character varying(30)  |
```

该视图包含以下信息:

f_table_catalog, f_table_schema, f_table_name 指定了表、视图、索引、函数、操作符、数据类型、以及元数据的名称。" f_table_catalog " 指定了表、视图、索引、函数、操作符、数据类型、以及元数据的名称。" f_table_schema " 指定了表、视图、索引、函数、操作符、数据类型、以及元数据的名称。" f_table_name " 指定了表、视图、索引、函数、操作符、数据类型、以及元数据的名称。PostgreSQL 数据库系统必须实现该视图。" f_table_catalog " 指定了表、视图、索引、函数、操作符、数据类型、以及元数据的名称。" f_table_schema " 指定了表、视图、索引、函数、操作符、数据类型、以及元数据的名称。" f_table_name " 指定了表、视图、索引、函数、操作符、数据类型、以及元数据的名称。PostgreSQL 数据库系统必须实现该视图。

f_geometry_column 指定了表的名称。

coord_dimension 指定了表的维数 (2, 3, 或 4 维)。

srid 指定了表的 SRID 值。ID 值, SPATIAL_REF_SYS 值 (foreign key)。

type 指定了表的几何数据类型。POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION, XYM, POINTM, LINESTRINGM, POLYGONM, MULTIPOINTM, MULTILINESTRINGM, MULTIPOLYGONM, GEOMETRYCOLLECTIONM。"GEOMETRY" 指定了表的几何数据类型。

4.6.3 geometry_columns 视图

AddGeometryColumn() 函数用于向数据库中添加新的几何数据类型, SQL 语句 (bulk insert) 语句。该函数, 向 geometry_columns 视图添加新的记录。PostGIS 2.0 版本, 该函数 typmod 参数用于指定新的几何数据类型。

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

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

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

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

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

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

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

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

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

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

PSQL 命令:

```
\d pois_ny;
```

显示表结构。使用 `typmod` 参数。

Table "public.pois_ny"

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

Indexes:

"pois_ny_pkey" PRIMARY KEY, btree (gid)

Check constraints:

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

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

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

4.7.1 SQL 脚本文件

PostGIS 提供了将 SQL 脚本文件 (formatted) SQL 脚本文件。Oracle 风格的 SQL 脚本文件, SQL 脚本文件 SQL "INSERT" 脚本文件 (piping) 脚本文件。

脚本文件 (roads.sql) 脚本文件:

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

"psql" SQL 脚本文件 PostgreSQL 脚本文件。

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

4.7.2 shp2pgsql: ESRI shapefile 脚本文件

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

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

c|a|d|p -- 脚本文件:

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

-? 脚本文件。

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

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

-k 脚本文件 (脚本, 脚本) 脚本文件。shapefile 脚本文件。

4.8 地理函数

SQL 使用 shapefile 格式/地理数据库格式/地理数据库格式。SQL 使用地理数据库格式/地理数据库格式/地理数据库格式。

4.8.1 SQL 地理数据库格式

地理数据库格式/地理数据库格式/地理数据库格式 SQL 使用 (select) 地理数据库格式/地理数据库格式/地理数据库格式。

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

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

(6 rows)

地理数据库格式/地理数据库格式/地理数据库格式。地理数据库格式/地理数据库格式/地理数据库格式。SQL 使用地理数据库格式/地理数据库格式/地理数据库格式。

ST_Intersects This function tells whether two geometries share any space.

= 地理数据库格式/地理数据库格式/地理数据库格式。地理数据库格式/地理数据库格式/地理数据库格式。'POLYGON((0 0,1 1,1 0,0 0))' 地理数据库格式/地理数据库格式/地理数据库格式 (地理数据库格式)。

地理数据库格式/地理数据库格式/地理数据库格式。SQL 使用地理数据库格式/地理数据库格式/地理数据库格式, "ST_GeomFromText()" 地理数据库格式/地理数据库格式/地理数据库格式。地理数据库格式/地理数据库格式/地理数据库格式 312 地理数据库格式/地理数据库格式/地理数据库格式。

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

地理数据库格式/地理数据库格式/地理数据库格式"ROADS_GEOM" 地理数据库格式/地理数据库格式/地理数据库格式。

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((...))');
```

地理数据库格式/地理数据库格式/地理数据库格式" (map frame)" 地理数据库格式/地理数据库格式/地理数据库格式" 地理数据库格式 (frame-based)" 地理数据库格式/地理数据库格式/地理数据库格式。

"&&" 地理数据库格式/地理数据库格式/地理数据库格式 BOX3D 地理数据库格式/地理数据库格式/地理数据库格式。地理数据库格式/地理数据库格式/地理数据库格式, 地理数据库格式/地理数据库格式/地理数据库格式。

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);
```

地理数据库格式/地理数据库格式/地理数据库格式 SRID 312 地理数据库格式/地理数据库格式/地理数据库格式。

4.8.2 使用pgsql2shp

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

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

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

选项如下：

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

4.9 索引

PostgreSQL 支持多种索引方法，包括 B-Tree、R-Tree、GiST 和 BRIN。B-Tree 索引适用于属性数据，而 R-Tree、GiST 和 BRIN 索引适用于空间数据。

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

- GiST (Generalized Search Tree) 是一种用于空间数据的索引方法。它支持多种几何类型，如点、线、面等。PostGIS 使用 GiST 索引来加速空间查询。R-Tree 也是一种用于空间数据的索引方法，它通过存储几何对象的边界框来加速查询。
- **BRIN (Block Range Index)** indexes operate by summarizing the spatial extent of ranges of table records. Search is done via a scan of the ranges. BRIN is only appropriate for use for some kinds of data (spatially sorted, with infrequent or no update). But it provides much faster index create time, and much smaller index size.

- **SP-GiST (Space-Partitioned Generalized Search Tree)** is a generic index method that supports partitioned search trees such as quad-trees, k-d trees, and radix trees (tries).

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

GiST “[Indexing Geometric Objects](#)” is a generic index method, [Indexing Geometric Objects](#). GIS [Indexing Geometric Objects](#), [Indexing Geometric Objects](#) B-Tree [Indexing Geometric Objects](#) (tries, [Indexing Geometric Objects](#)) [Indexing Geometric Objects](#) GiST [Indexing Geometric Objects](#).

GIS [Indexing Geometric Objects](#), [Indexing Geometric Objects](#) (tries, [Indexing Geometric Objects](#)). [Indexing Geometric Objects](#).

“[Indexing Geometric Objects](#)” [Indexing Geometric Objects](#):

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

[Indexing Geometric Objects](#) 2D [Indexing Geometric Objects](#). [Indexing Geometric Objects](#) PostgreSQL 2.0 [Indexing Geometric Objects](#) n [Indexing Geometric Objects](#), [Indexing Geometric Objects](#):

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

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

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

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

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

4.9.2 GiST

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

GIS [Indexing Geometric Objects](#), [Indexing Geometric Objects](#) (tries, [Indexing Geometric Objects](#)). [Indexing Geometric Objects](#).

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 . PostGIS 2.0 n ,:

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

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

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

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

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

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

” ” GiST ::

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

2D . PostGIS 2.0 n ,:

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

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

To summarize using BRIN for spatial data:

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

4.9.3 GiST 索引

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

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

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

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

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

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

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

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

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

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

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

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

There is no support for kNN searches at the moment.

4.9.4 性能调优

PostgreSQL 的 GiST 索引在性能调优方面有一些需要注意的地方。首先，确保索引的统计信息是最新的，这可以通过 `VACUUM ANALYZE` 来实现。其次，检查查询计划，确保索引被正确使用。最后，考虑使用 `SET ENABLE_SEQSCAN=OFF` 来禁用顺序扫描，这可以提高某些查询的性能。

以下是一些性能调优的建议：

- 检查查询计划并确认你的查询实际上计算了你需要的东西。一个错误的 JOIN，无论是忘记还是连接到错误的表，都可能意外地多次检索表记录。要获取查询计划，请在查询前执行 `EXPLAIN`。
- 确保统计信息已收集，以提供查询规划器有关表中值的数量和分布的更好信息，以便做出有关索引使用的决策。**VACUUM ANALYZE** 将同时计算两者。

你应该定期对你的数据库进行真空操作。许多 PostgreSQL DBAs 运行 **VACUUM** 作为离峰时间的 cron 作业，定期进行。

- 对于某些查询，禁用顺序扫描（`SET ENABLE_SEQSCAN=OFF`）可能有所帮助。这可以通过在会话级别或全局级别设置来实现。例如，在会话级别，你可以使用 `SET ENABLE_SEQSCAN=OFF;`。在全局级别，你可以修改 `postgresql.conf` 文件中的 `enable_seqscan` 参数。默认情况下，该参数设置为 `on`。如果你希望禁用顺序扫描，可以将其设置为 `off`。
- 对于某些查询，调整 `random_page_cost` 参数可能有所帮助。该参数位于 `postgresql.conf` 文件中，默认值为 4。如果你希望 planner 更倾向于使用索引而不是进行随机访问，可以降低该值。例如，将其设置为 1 或 2。
- 如果 `SET ENABLE_SEQSCAN TO OFF;` 没有帮助，你的查询可能正在使用 SQL 构造，这是 Postgres 规划器目前无法优化的。可能是可以重写查询，使其更容易处理。例如，带有内联 SELECT 的子查询可能无法产生高效的计划，但可能可以通过使用 LATERAL JOIN 来重写。

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

Chapter 5

Spatial Queries

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

5.1 Determining Spatial Relationships

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

5.1.1 Dimensionally Extended 9-Intersection Model

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

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

Boundary

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

Interior

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

Exterior

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

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

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

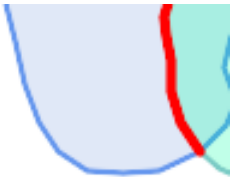
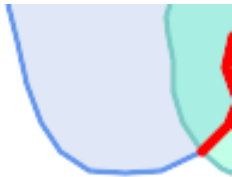
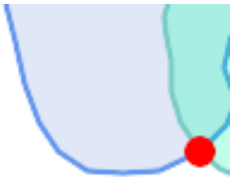
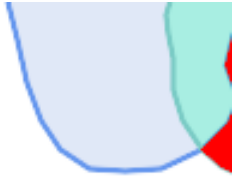
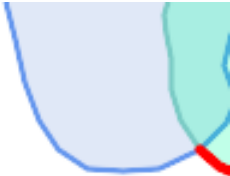
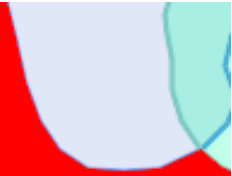
- 0 => point
- 1 => line
- 2 => area
- F => empty set

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

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

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



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

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

For more information, refer to:

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

5.1.2 Named Spatial Relationships

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

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


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

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

5.1.3 General Spatial Relationships

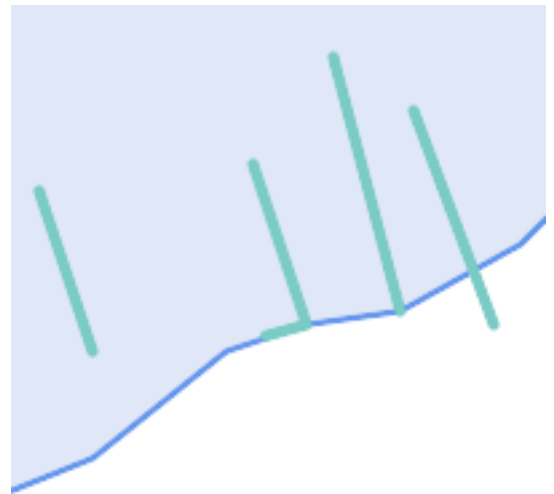
In some cases the named spatial relationships are insufficient to provide a desired spatial filter condition.



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

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

Clearly, a simpler and faster solution is desirable.



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

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

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

```
SELECT ST_Relate( 'LINESTRING (1 1, 5 5)',
                  'POLYGON ((3 3, 3 7, 7 7, 7 3, 3 3))' );
st_relate
-----
1010F0212
```

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

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

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

```
-- Find road segments that intersect in a line
SELECT a.id
```

```
FROM roads a, roads b
WHERE a.id != b.id
      AND a.geom && b.geom
      AND ST_Relate(a.geom, b.geom, '1*1***1**');
```

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

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

5.2 Using Spatial Indexes

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

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

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

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

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

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

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

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

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

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

5.3 Examples of Spatial SQL

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

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

The definition of the `bc_municipality` table is:

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

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

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

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

km_roads

70842.1243039643

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

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

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

hectares

32657.9103824927

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

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

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

name	hectares
-----	-----
TUMBLER RIDGE	155020.02556131

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

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

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

```
SELECT
  m.name,
  sum(ST_Length(r.geom))/1000 as roads_km
FROM bc_roads AS r
JOIN bc_municipality AS m
  ON ST_Contains(m.geom, r.geom)
GROUP BY m.name
ORDER BY roads_km;
```

name	roads_km
SURREY	1539.47553551242
VANCOUVER	1450.33093486576
LANGLEY DISTRICT	833.793392535662
BURNABY	773.769091404338
PRINCE GEORGE	694.37554369147
...	

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

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

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

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

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

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

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

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

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

gid	name	area
12	SPALLUMCHEEN	257374619.430216

Chapter 6

安装与使用

6.1 安装与使用

6.1.1 安装

PostgreSQL (8.0 版本) 的 optimizer 对 TOAST 支持不够完善。TOAST 支持不够完善 (即, 不支持 TOAST 的 extension room) 的 (即, 不支持 TOAST 的 extension room) 的。请参考 [the PostgreSQL Documentation for TOAST](#) 文档。

PostgreSQL (8.0 版本) 的 optimizer 对 TOAST 支持不够完善。TOAST 支持不够完善 (即, 不支持 TOAST 的 extension room) 的 (即, 不支持 TOAST 的 extension room) 的。请参考 [the PostgreSQL Documentation for TOAST](#) 文档。

PostgreSQL (8.0 版本) 的 optimizer 对 TOAST 支持不够完善。TOAST 支持不够完善 (即, 不支持 TOAST 的 extension room) 的 (即, 不支持 TOAST 的 extension room) 的。请参考 [the PostgreSQL Documentation for TOAST](#) 文档。

PostgreSQL (8.0 版本) 的 optimizer 对 TOAST 支持不够完善。TOAST 支持不够完善 (即, 不支持 TOAST 的 extension room) 的 (即, 不支持 TOAST 的 extension room) 的。请参考 [the PostgreSQL Documentation for TOAST](#) 文档。

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

6.1.2 使用

PostgreSQL 对 TOAST 支持不够完善。TOAST 支持不够完善 (即, 不支持 TOAST 的 extension room) 的 (即, 不支持 TOAST 的 extension room) 的。请参考 [the PostgreSQL Documentation for TOAST](#) 文档。

PostgreSQL 对 TOAST 支持不够完善。TOAST 支持不够完善 (即, 不支持 TOAST 的 extension room) 的 (即, 不支持 TOAST 的 extension room) 的。请参考 [the PostgreSQL Documentation for TOAST](#) 文档。

PostgreSQL 对 TOAST 支持不够完善。TOAST 支持不够完善 (即, 不支持 TOAST 的 extension room) 的 (即, 不支持 TOAST 的 extension room) 的。请参考 [the PostgreSQL Documentation for TOAST](#) 文档。

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

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

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

`mytable`, mytable `trigger` `bbox` " " `bbox` `UPDATE`

6.2 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒

`CREATE CLUSTER postgresql ON DISKGROUP 'diskgroup1' USING SYMLINKS;
ALTER CLUSTER postgresql ADD NODE 'node1';
ALTER CLUSTER postgresql ADD NODE 'node2';`

1. PostgreSQL 12.1 PostGIS 3.0.0 GiST 3.11.0. 2. GiST 3.11.0 NULL 3.11.0. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36. 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53. 54. 55. 56. 57. 58. 59. 60. 61. 62. 63. 64. 65. 66. 67. 68. 69. 70. 71. 72. 73. 74. 75. 76. 77. 78. 79. 80. 81. 82. 83. 84. 85. 86. 87. 88. 89. 90. 91. 92. 93. 94. 95. 96. 97. 98. 99. 100. 101. 102. 103. 104. 105. 106. 107. 108. 109. 110. 111. 112. 113. 114. 115. 116. 117. 118. 119. 120. 121. 122. 123. 124. 125. 126. 127. 128. 129. 130. 131. 132. 133. 134. 135. 136. 137. 138. 139. 140. 141. 142. 143. 144. 145. 146. 147. 148. 149. 150. 151. 152. 153. 154. 155. 156. 157. 158. 159. 160. 161. 162. 163. 164. 165. 166. 167. 168. 169. 170. 171. 172. 173. 174. 175. 176. 177. 178. 179. 180. 181. 182. 183. 184. 185. 186. 187. 188. 189. 190. 191. 192. 193. 194. 195. 196. 197. 198. 199. 200. 201. 202. 203. 204. 205. 206. 207. 208. 209. 210. 211. 212. 213. 214. 215. 216. 217. 218. 219. 220. 221. 222. 223. 224. 225. 226. 227. 228. 229. 230. 231. 232. 233. 234. 235. 236. 237. 238. 239. 240. 241. 242. 243. 244. 245. 246. 247. 248. 249. 250. 251. 252. 253. 254. 255. 256. 257. 258. 259. 260. 261. 262. 263. 264. 265. 266. 267. 268. 269. 270. 271. 272. 273. 274. 275. 276. 277. 278. 279. 280. 281. 282. 283. 284. 285. 286. 287. 288. 289. 290. 291. 292. 293. 294. 295. 296. 297. 298. 299. 300. 301. 302. 303. 304. 305. 306. 307. 308. 309. 310. 311. 312. 313. 314. 315. 316. 317. 318. 319. 320. 321. 322. 323. 324. 325. 326. 327. 328. 329. 330. 331. 332. 333. 334. 335. 336. 337. 338. 339. 340. 341. 342. 343. 344. 345. 346. 347. 348. 349. 350. 351. 352. 353. 354. 355. 356. 357. 358. 359. 360. 361. 362. 363. 364. 365. 366. 367. 368. 369. 370. 371. 372. 373. 374. 375. 376. 377. 378. 379. 380. 381. 382. 383. 384. 385. 386. 387. 388. 389. 390. 391. 392. 393. 394. 395. 396. 397. 398. 399. 400. 401. 402. 403. 404. 405. 406. 407. 408. 409. 410. 411. 412. 413. 414. 415. 416. 417. 418. 419. 420. 421. 422. 423. 424. 425. 426. 427. 428. 429. 430. 431. 432. 433. 434. 435. 436. 437. 438. 439. 440. 441. 442. 443. 444. 445. 446. 447. 448. 449. 450. 451. 452. 453. 454. 455. 456. 457. 458. 459. 460. 461. 462. 463. 464. 465. 466. 467. 468. 469. 470. 471. 472. 473. 474. 475. 476. 477. 478. 479. 480. 481. 482. 483. 484. 485. 486. 487. 488. 489. 490. 491. 492. 493. 494. 495. 496. 497. 498. 499. 500. 501. 502. 503. 504. 505. 506. 507. 508. 509. 510. 511. 512. 513. 514. 515. 516. 517. 518. 519. 520. 521. 522. 523. 524. 525. 526. 527. 528. 529. 530. 531. 532. 533. 534. 535. 536. 537. 538. 539. 540. 541. 542. 543. 544. 545. 546. 547. 548. 549. 550. 551. 552. 553. 554. 555. 556. 557. 558. 559. 560. 561. 562. 563. 564. 565. 566. 567. 568. 569. 570. 571. 572. 573. 574. 575. 576. 577. 578. 579. 580. 581. 582. 583. 584. 585. 586. 587. 588. 589. 590. 591. 592. 593. 594. 595. 596. 597. 598. 599. 600. 601. 602. 603. 604. 605. 606. 607. 608. 609. 610. 611. 612. 613. 614. 615. 616. 617. 618. 619. 620. 621. 622. 623. 624. 625. 626. 627. 628. 629. 630. 631. 632. 633. 634. 635. 636. 637. 638. 639. 640. 641. 642. 643. 644. 645. 646. 647. 648. 649. 650. 651. 652. 653. 654. 655. 656. 657. 658. 659. 660. 661. 662. 663. 664. 665. 666. 667. 668. 669. 670. 671. 672. 673. 674. 675. 676. 677. 678. 679. 680. 681. 682. 683. 684. 685. 686. 687. 688. 689. 690. 691. 692. 693. 694. 695. 696. 697. 698. 699. 700. 701. 702. 703. 704. 705. 706. 707. 708. 709. 710. 711. 712. 713. 714. 715. 716. 717. 718. 719. 720. 721. 722. 723. 724. 725. 726. 727. 728. 729. 730. 731. 732. 733. 734. 735. 736. 737. 738. 739. 740. 741. 742. 743. 744. 745. 746. 747. 748. 749. 750. 751. 752. 753. 754. 755. 756. 757. 758. 759. 760. 761. 762. 763. 764. 765. 766. 767. 768. 769. 770. 771. 772. 773. 774. 775. 776. 777. 778. 779. 780. 781. 782. 783. 784. 785. 786. 787. 788. 789. 790. 791. 792. 793. 794. 795. 796. 797. 798. 799. 800. 801. 802. 803. 804. 805. 806. 807. 808. 809. 810. 811. 812. 813. 814. 815. 816. 817. 818. 819. 820. 821. 822. 823. 824. 825. 826. 827. 828. 829. 830. 831. 832.

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

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

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

[illegible]

6.3

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

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

```
AddGeometryColumn('public','geometry_columns', 'geometry_columns'
    geometry_columns
```

`WHERE geom IS NOT NULL AND ST_IsEmpty(geom)=0`

Chapter 7

PostGIS Reference

PostGIS 2.2.3, PostGIS 2.2.3
PostGIS 2.2.3.

Note

[illegible]

7.1 PostgreSQL PostGIS Geometry/Geography/Box

7.1.1 box2d

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



```
box3d [REDACTED] postgres [REDACTED]. ST_3DExtent  
[REDACTED] box3d [REDACTED].
```

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).

□ □ □ □ □

[illegible]

box3d	box3d
geometry	geometry

几何

Section 13.7

7.1.2 box3d

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

几何

box3d 数据类型在 PostGIS 3.6.1 版本中引入。ST_3DExtent 函数返回 box3d 类型的值。

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).

数据类型

数据类型在 PostGIS 3.6.1 版本中引入。

数据类型	几何
box	几何
box2d	几何
geometry	几何

几何

Section 13.7

7.1.3 geometry

geometry — geography 数据类型在 PostGIS 3.6.1 版本中引入。

几何

geography 数据类型在 PostGIS 3.6.1 版本中引入。

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

数据类型

数据类型在 PostGIS 3.6.1 版本中引入。

数据类型	几何
box	几何
box2d	几何
box3d	几何
bytea	几何

geography	几何
text	文本

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 地理数据类型。

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

地理数据类型

地理数据类型。

地理数据类型	几何
geometry	几何

❏

Section 4.3, Section 4.3

7.2 几何函数

7.2.1 AddGeometryColumn

AddGeometryColumn — 添加几何列。

Synopsis

text **AddGeometryColumn**(varchar table_name, varchar column_name, integer srid, varchar type, integer dimension, boolean use_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);

❏

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

Note



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

Note



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



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



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



This method supports Circular Strings and Curves.

❏: 2.0.0 ❏. use_typmod ❏. ❏ typmod ❏
❏.

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

-- Describing the table shows a simple table with a single "id" column.
postgis=# \d my_schema.my_spatial_table
                                Table "my_schema.my_spatial_table"
Column | Type      | Modifiers
-----+-----+-----
 id    | integer   | not null default nextval('my_schema.my_spatial_table_id_seq'::regclass)

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

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

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

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

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

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

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

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

❏

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

7.2.2 DropGeometryColumn

DropGeometryColumn — 从 geometry_columns 表中删除列。

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`
`column_name);`

- ✔ 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

❏: 2.0.0 版。从 geometry_columns 表中删除列。从 geometry_columns 表中删除列，使用 ALTER TABLE 删除列。

❏

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

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

❏

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

7.2.3 DropGeometryTable

DropGeometryTable — 从 geometry_columns 表中删除表。

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);
```

注意

`geometry_columns` 表是 schema-aware 的。因此，在 PostgreSQL 12 之前，使用 `current_schema()` 函数来指定 schema 名称。



Note

PostGIS 2.0.0 之前，`DropGeometryTable` 函数只接受表名。从 2.0.0 开始，`DropGeometryTable` 函数接受 schema 名称和表名。因此，在 PostgreSQL 12 之前，使用 `current_schema()` 函数来指定 schema 名称。

示例

```
SELECT DropGeometryTable ('my_schema','my_spatial_table');
----RESULT output ---
my_schema.my_spatial_table dropped.

-- The above is now equivalent to --
DROP TABLE my_schema.my_spatial_table;
```

注意

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

7.2.4 Find_SRID

Find_SRID — Returns the SRID defined for a geometry column.

Synopsis

```
integer Find_SRID(varchar a_schema_name, varchar a_table_name, varchar a_geomfield_name);
```

注意

Returns the integer SRID of the specified geometry column by searching through the `GEOMETRY_COLUMNS` table. If the geometry column has not been properly added (e.g. with the [AddGeometryColumn](#) function), this function will not work.

示例

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

¶¶

ST_SRID

7.2.5 Populate_Geometry_Columns

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

Synopsis

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

¶¶

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

如果 use_typmod 为 true，则函数将尝试为所有几何列添加类型修饰符。如果 use_typmod 为 false，则函数将尝试为所有几何列添加适当的空间约束。该函数将返回一个整数，表示成功更新的列数。如果返回值为 3，则表示所有列都已成功更新。如果返回值为 0，则表示没有列需要更新。

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

oid 参数是几何列的 OID，SRID 是空间参考 ID。如果 SRID 为 0，则函数将尝试为所有几何列添加适当的空间约束。如果 SRID 不为 0，则函数将尝试为所有几何列添加类型修饰符。

oid 参数是几何列的 OID，geometry_columns 是几何列的表名。如果 geometry_columns 为 0，则函数将尝试为所有几何列添加适当的空间约束。如果 geometry_columns 不为 0，则函数将尝试为所有几何列添加类型修饰符。

该函数将返回一个整数，表示成功更新的列数。如果返回值为 3，则表示所有列都已成功更新。如果返回值为 0，则表示没有列需要更新。该函数将尝试为所有几何列添加适当的空间约束。如果 use_typmod 为 true，则函数将尝试为所有几何列添加类型修饰符。

1.4.0 版本引入。

版本 2.0.0 中，该函数将尝试为所有几何列添加类型修饰符。如果 use_typmod 为 true，则函数将尝试为所有几何列添加类型修饰符。如果 use_typmod 为 false，则函数将尝试为所有几何列添加适当的空间约束。

版本 2.0.0 中，该函数将尝试为所有几何列添加类型修饰符。如果 use_typmod 为 true，则函数将尝试为所有几何列添加类型修饰符。如果 use_typmod 为 false，则函数将尝试为所有几何列添加适当的空间约束。


```
CREATE TABLE public.myspatial_table(gid serial, geom geometry);
INSERT INTO myspatial_table(geom) VALUES(ST_GeomFromText('LINESTRING(1 2, 3 4)',4326) );
-- This will now use typ modifiers.  For this to work, there must exist data
SELECT Populate_Geometry_Columns('public.myspatial_table'::regclass);

populate_geometry_columns
-----
1

\d myspatial_table

Table "public.myspatial_table"
Column |          Type          | Modifiers
-----+-----+-----
gid    | integer                | not null default nextval('myspatial_table_gid_seq'::regclass)
geom   | geometry(LineString,4326) |

-- This will change the geometry columns to use constraints if they are not typmod or have constraints already.
--For this to work, there must exist data
CREATE TABLE public.myspatial_table_cs(gid serial, geom geometry);
INSERT INTO myspatial_table_cs(geom) VALUES(ST_GeomFromText('LINESTRING(1 2, 3 4)',4326) );
SELECT Populate_Geometry_Columns('public.myspatial_table_cs'::regclass, false);
populate_geometry_columns
-----
1

\d myspatial_table_cs

Table "public.myspatial_table_cs"
Column |  Type  | Modifiers
-----+-----+-----
gid    | integer | not null default nextval('myspatial_table_cs_gid_seq'::regclass)
geom   | geometry |
Check constraints:
    "enforce_dims_geom" CHECK (st_ndims(geom) = 2)
    "enforce_geotype_geom" CHECK (geometrytype(geom) = 'LINESTRING'::text OR geom IS NULL)
    "enforce_srid_geom" CHECK (st_srid(geom) = 4326)
```

7.2.6 UpdateGeometrySRID

UpdateGeometrySRID — Updates the SRID of all features in a geometry column, and the table meta-data.

Synopsis

```
text UpdateGeometrySRID(varchar table_name, varchar column_name, integer srid);
text UpdateGeometrySRID(varchar schema_name, varchar table_name, varchar column_name, integer srid);
text UpdateGeometrySRID(varchar catalog_name, varchar schema_name, varchar table_name, varchar column_name, integer srid);
```

更新

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



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



This method supports Circular Strings and Curves.

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 列的 SRID 为 4326 的表:

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

更新 DDL 语句:

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

更新 (未知) 的 SRID 列的 SRID 为 3857 的表, DDL 语句: 在 PostgreSQL 中, 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 几何构造器 (constructor)

7.3.1 ST_Collect

ST_Collect — Creates a GeometryCollection or Multi* geometry from a set of geometries.

Synopsis

```
geometry ST_Collect(geometry g1, geometry g2);
geometry ST_Collect(geometry[] g1_array);
geometry ST_Collect(geometry set g1field);
```

ST_Collect

Collects geometries into a geometry collection. The result is either a Multi* or a GeometryCollection, depending on whether the input geometries have the same or different types (homogeneous or heterogeneous). The input geometries are left unchanged within the collection.

Variant 1: accepts two input geometries

Variant 2: accepts an array of geometries

Variant 3: aggregate function accepting a rowset of geometries.



Note

If any of the input geometries are collections (Multi* or GeometryCollection) ST_Collect returns a GeometryCollection (since that is the only type which can contain nested collections). To prevent this, use **ST_Dump** in a subquery to expand the input collections to their atomic elements (see example below).



Note

ST_Collect and **ST_Union** appear similar, but in fact operate quite differently. ST_Collect aggregates geometries into a collection without changing them in any way. ST_Union geometrically merges geometries where they overlap, and splits linestrings at intersections. It may return single geometries when it dissolves boundaries.

1.4.0 添加了对 3D 的支持。ST_MakeLine 现在支持 3D 点。ST_MakeLine 现在支持 3D 点。



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



This method supports Circular Strings and Curves.

ST_Collect: XLink

Collect 2D points.

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

st_astext

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

Collect 3D points.

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

st_asewkt

MULTIPOINT(1 2 3,1 2 4)

Collect curves.

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

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

例: 聚合几何

Using an array constructor for a subquery.

```
SELECT ST_Collect( ARRAY( SELECT geom FROM sometable ) );
```

Using an array constructor for values.

```
SELECT ST_AsText( ST_Collect(
    ARRAY[ ST_GeomFromText('LINESTRING(1 2, 3 4)'),
          ST_GeomFromText('LINESTRING(3 4, 4 5)') ] ) ) As wktcollect;

--wkt collect --
MULTILINESTRING((1 2,3 4),(3 4,4 5))
```

例: 聚合几何

Creating multiple collections by grouping geometries in a table.

```
SELECT stusps, ST_Collect(f.geom) as geom
    FROM (SELECT stusps, (ST_Dump(geom)).geom As geom
          FROM
            somestatetable ) As f
    GROUP BY stusps
```

例:

ST_Dump, ST_AsBinary

7.3.2 ST_LineFromMultiPoint

ST_LineFromMultiPoint — 从多点点集创建线。

Synopsis

geometry **ST_LineFromMultiPoint**(geometry aMultiPoint);

例:

从多点点集创建线。

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



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

例

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

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

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

例

[ST_AsEWKT](#), [ST_AsKML](#)

7.3.3 ST_MakeEnvelope

ST_MakeEnvelope — 根据给定的边界框和SRID创建几何图形。SRID 指定 SRS 标识符。

Synopsis

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

例

根据给定的边界框和SRID创建几何图形。SRID 指定 SRS 标识符。

1.5 版本之前。

2.0 版本 SRID 参数 (envelope) 是可选的。

例: 创建多边形

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

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

例

[ST_MakePoint](#), [ST_MakePoint](#), [ST_Point](#), [ST_SRID](#)

7.3.4 ST_MakeLine

ST_MakeLine — 根据给定的点集创建几何图形。

Synopsis

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

概要

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

Variant 1: accepts two input geometries

Variant 2: accepts an array of geometries

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

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



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

2.0.0 函数返回的 LineString 几何体将包含所有输入几何体的点。

2.0.0 函数返回的 LineString 几何体将包含所有输入几何体的点。

1.4.0 函数返回的 LineString 几何体将包含所有输入几何体的点。在 2.0.0 之前，ST_MakeLine 函数返回的 LineString 几何体将包含所有输入几何体的点。在 2.0.0 之前，ST_MakeLine 函数返回的 LineString 几何体将包含所有输入几何体的点。

例: 创建一条由两个点组成的线。

Create a line composed of two points.

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

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

3D 点组成的 2D 盒子 BOX3D 函数。

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

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

使用 ST_MakeLine 函数创建 LineString 几何体。

```
select ST_AsText( ST_MakeLine( 'LINESTRING(0 0, 1 1)', 'LINESTRING(2 2, 3 3)' ) );
```

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

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) ] ) );

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

名称: ST_MakeLine

给定 GPS 点集，ST_MakeLine 函数返回由这些点组成的 LineString。GPS 点集可以是点数组或子查询结果。

Using aggregate ORDER BY provides a correctly-ordered LineString.

```
SELECT gps.track_id, ST_MakeLine(gps.geom ORDER BY gps_time) As geom
FROM gps_points As gps
GROUP BY track_id;
```

Prior to PostgreSQL 9, ordering in a subquery can be used. However, sometimes the query plan may not respect the order of the subquery.

```
SELECT gps.track_id, ST_MakeLine(gps.geom) As geom
FROM ( SELECT track_id, gps_time, geom
      FROM gps_points ORDER BY track_id, gps_time ) As gps
GROUP BY track_id;
```

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.

☐☐

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

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

-- Create a 3D point (e.g. has altitude)
SELECT ST_MakePoint(1, 2,1.5);

-- Get z of point
SELECT ST_Z(ST_MakePoint(1, 2,1.5));
result
-----
1.5
```

☐☐

ST_GeomFromText, **ST_PointFromText**, **ST_SetSRID**, **ST_MakePointM**, **ST_Point**, **ST_PointZ**, **ST_PointM**, **ST_PointZM**

7.3.6 ST_MakePointM

ST_MakePointM — x, y ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐.

Synopsis

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

点

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.

点



Note

[ST_AsEWKT](#) is used for text output because [ST_AsText](#) does not support M values.

Create point with unknown SRID.

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

```

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

x, y 和 measure 值。

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

```

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

Get measure of created point.

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

```

result
-----
10
```

点

[ST_MakePoint](#), [ST_SetSRID](#), [ST_PointM](#), [ST_PointZM](#)

7.3.7 ST_MakePolygon

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

Synopsis

geometry **ST_MakePolygon**(geometry linestring);

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

返回

返回类型 (shell) 是 `geometry`。返回类型是 `geometry`。

Variant 1: Accepts one shell LineString.

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



Note

此函数在 PostgreSQL 12 之前版本中不可用。在 PostgreSQL 12 及更高版本中，使用 `ST_LineMerge` 和 `ST_Dump` 可以实现类似功能。



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

示例: 创建多边形

以下 SQL 语句将 LineString 转换为 Polygon:

```
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;
```

以下 SQL 语句将 LineString 转换为 Polygon:

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

st_asewkt

```
POLYGON((75.15 29.53 1,77 29 1,77.6 29.5 1,75.15 29.53 1))
```

Create a Polygon from a LineString with measures

```
SELECT ST_AsEWKT( ST_MakePolygon( 'LINESTRINGM(75.15 29.53 1,77 29 1,77.6 29.5 2, 75.15
29.53 2)') ));
```

st_asewkt

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

注意: 使用 `ST_MakePolygon` 时

必须使用 `ST_IsClosed` 函数。

```
SELECT ST_MakePolygon( ST_ExteriorRing( ST_Buffer(ring.line,10)),
    ARRAY[ ST_Translate(ring.line, 1, 1),
          ST_ExteriorRing(ST_Buffer(ST_Point(20,20),1)) ]
    )
FROM (SELECT ST_ExteriorRing(
    ST_Buffer(ST_Point(10,10),10,10)) AS line ) AS ring;
```

Create a set of province boundaries with holes representing lakes. The input is a table of province Polygons/MultiPolygons and a table of water linestrings. Lines forming lakes are determined by using [ST_IsClosed](#). The province linework is extracted by using [ST_Boundary](#). As required by `ST_MakePolygon`, the boundary is forced to be a single LineString by using [ST_LineMerge](#). (However, note that if a province has more than one region or has islands this will produce an invalid polygon.) Using a LEFT JOIN ensures all provinces are included even if they have no lakes.



Note

NULL 值在 `ST_MakePolygon` 函数中会被忽略，因此使用 `CASE` 语句。

```
SELECT p.gid, p.province_name,
    CASE WHEN array_agg(w.geom) IS NULL
    THEN p.geom
    ELSE ST_MakePolygon( ST_LineMerge(ST_Boundary(p.geom)),
        array_agg(w.geom)) END
FROM
    provinces p LEFT JOIN waterlines w
        ON (ST_Within(w.geom, p.geom) AND ST_IsClosed(w.geom))
GROUP BY p.gid, p.province_name, p.geom;
```

Another technique is to utilize a correlated subquery and the `ARRAY()` constructor that converts a row set to an array.

```
SELECT p.gid, p.province_name,
    CASE WHEN EXISTS( SELECT w.geom
        FROM waterlines w
        WHERE ST_Within(w.geom, p.geom)
        AND ST_IsClosed(w.geom))
    THEN ST_MakePolygon(
        ST_LineMerge(ST_Boundary(p.geom)),
        ARRAY( SELECT w.geom
            FROM waterlines w
            WHERE ST_Within(w.geom, p.geom)
            AND ST_IsClosed(w.geom)))
    ELSE p.geom
    END AS geom
FROM provinces p;
```

函数

[ST_BuildArea](#) [ST_Polygon](#)

7.3.8 ST_Point

`ST_Point` — Creates a Point with X, Y and SRID values.

Synopsis

geometry **ST_Point**(float x, float y);

geometry **ST_Point**(float x, float y, integer srid=unknown);

返回

Returns a Point with the given X and Y coordinate values. This is the SQL-MM equivalent for **ST_MakePoint** that takes just X and Y.



Note

For geodetic coordinates, X is longitude and Y is latitude

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



This method implements the SQL/MM specification. SQL-MM 3: 6.1.2

示例:

```
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);
```

示例:

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;
```

参见

ST_MakePoint, ST_PointZ, ST_PointM, ST_PointZM, ST_SetSRID, ST_Transform

7.3.9 ST_PointZ

ST_PointZ — Creates a Point with X, Y, Z and SRID values.

Synopsis

geometry **ST_PointZ**(float x, float y, float z, integer srid=unknown);

返回

返回与 ST_Point 相同的类型。 ST_MakePoint 符合 OGC 规范。

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

示例

```
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)
```

参见

[ST_MakePoint](#), [ST_PointFromText](#), [ST_SetSRID](#), [ST_MakePointM](#)

7.3.10 ST_PointM

ST_PointM — Creates a Point with X, Y, M and SRID values.

Synopsis

geometry **ST_PointM**(float x, float y, float m, integer srid=unknown);

返回

返回与 ST_Point 相同的类型。 ST_MakePoint 符合 OGC 规范。

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

示例

```
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)
```

函数

[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);

函数

函数返回一个 `ST_Point` 类型的几何体。 `ST_MakePoint` 符合 OGC 标准。

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.

函数

```
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)
```

函数

[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);

函数

Returns a polygon built from the given LineString and sets the spatial reference system from the `srid`.

`ST_Polygon` is similar to [ST_MakePolygon](#) Variant 1 with the addition of setting the SRID.

, [ST_MakePoint](#), [ST_SetSRID](#)

**Note**

此函数在 PostGIS 3.6.1 版本中引入。它依赖于 `ST_LineMerge` 和 `ST_Dump` 函数。



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



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



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

创建 2D 多边形。

Create a 2D polygon.

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

-- result --

```
POLYGON((75 29, 77 29, 77 29, 75 29))
```

Create a 3D polygon.

```
SELECT ST_AsEWKT( ST_Polygon( ST_GeomFromEWKT('LINESTRING(75 29 1, 77 29 2, 77 29 3, 75 29 1)'), 4326) );
```

-- result --

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

函数

`ST_AsEWKT`, `ST_AsText`, `ST_GeomFromEWKT`, `ST_GeomFromText`, `ST_LineMerge`, `ST_MakePolygon`

7.3.13 ST_TileEnvelope

`ST_TileEnvelope` — Creates a rectangular Polygon in [Web Mercator](#) (SRID:3857) using the [XYZ tile system](#).

Synopsis

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

描述

Creates a rectangular Polygon giving the extent of a tile in the [XYZ tile system](#). The tile is specified by the zoom level Z and the XY index of the tile in the grid at that level. Can be used to define the tile bounds required by `ST_AsMVTGeom` to convert geometry into the MVT tile coordinate space.

By default, the tile envelope is in the [Web Mercator](#) coordinate system (SRID:3857) using the standard range of the Web Mercator system (-20037508.342789, 20037508.342789). This is the most common coordinate system used for MVT tiles. The optional bounds parameter can be used to generate tiles in

any coordinate system. It is a geometry that has the SRID and extent of the "Zoom Level zero" square within which the XYZ tile system is inscribed.

The optional `margin` parameter can be used to expand a tile by the given percentage. E.g. `margin=0.125` expands the tile by 12.5%, which is equivalent to `buffer=512` when the tile extent size is 4096, as used in [ST_AsMVTGeom](#). This is useful to create a tile buffer to include data lying outside of the tile's visible area, but whose existence affects the tile rendering. For example, a city name (a point) could be near an edge of a tile, so its label should be rendered on two tiles, even though the point is located in the visible area of just one tile. Using expanded tiles in a query will include the city point in both tiles. Use a negative value to shrink the tile instead. Values less than -0.5 are prohibited because that would eliminate the tile completely. Do not specify a margin when using with `ST_AsMVTGeom`. See the example for [ST_AsMVT](#).

例: 2.0.0 使用 SRID 4326.

2.1.0 使用 4326.

例: 使用 4326

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

 st_astext
-----
POLYGON((-10018754.1713945 0,-10018754.1713945 10018754.1713945,0 10018754.1713945,0  ←
0,-10018754.1713945 0))

SELECT ST_AsText( ST_TileEnvelope(3, 1, 1, ST_MakeEnvelope(-180, -90, 180, 90, 4326) ) );

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

例

[ST_MakeEnvelope](#)

7.3.14 ST_HexagonGrid

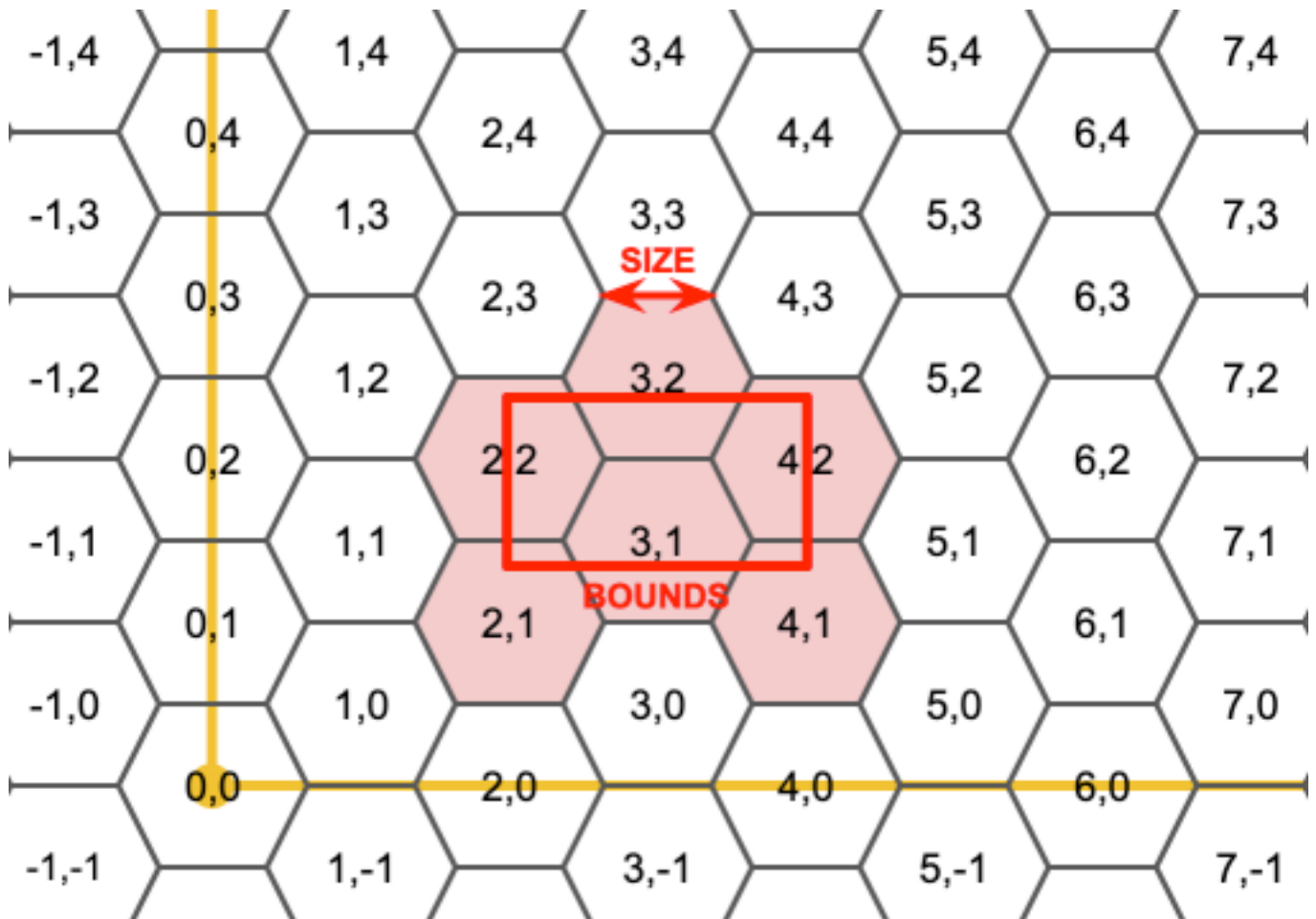
`ST_HexagonGrid` — Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.

Synopsis

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

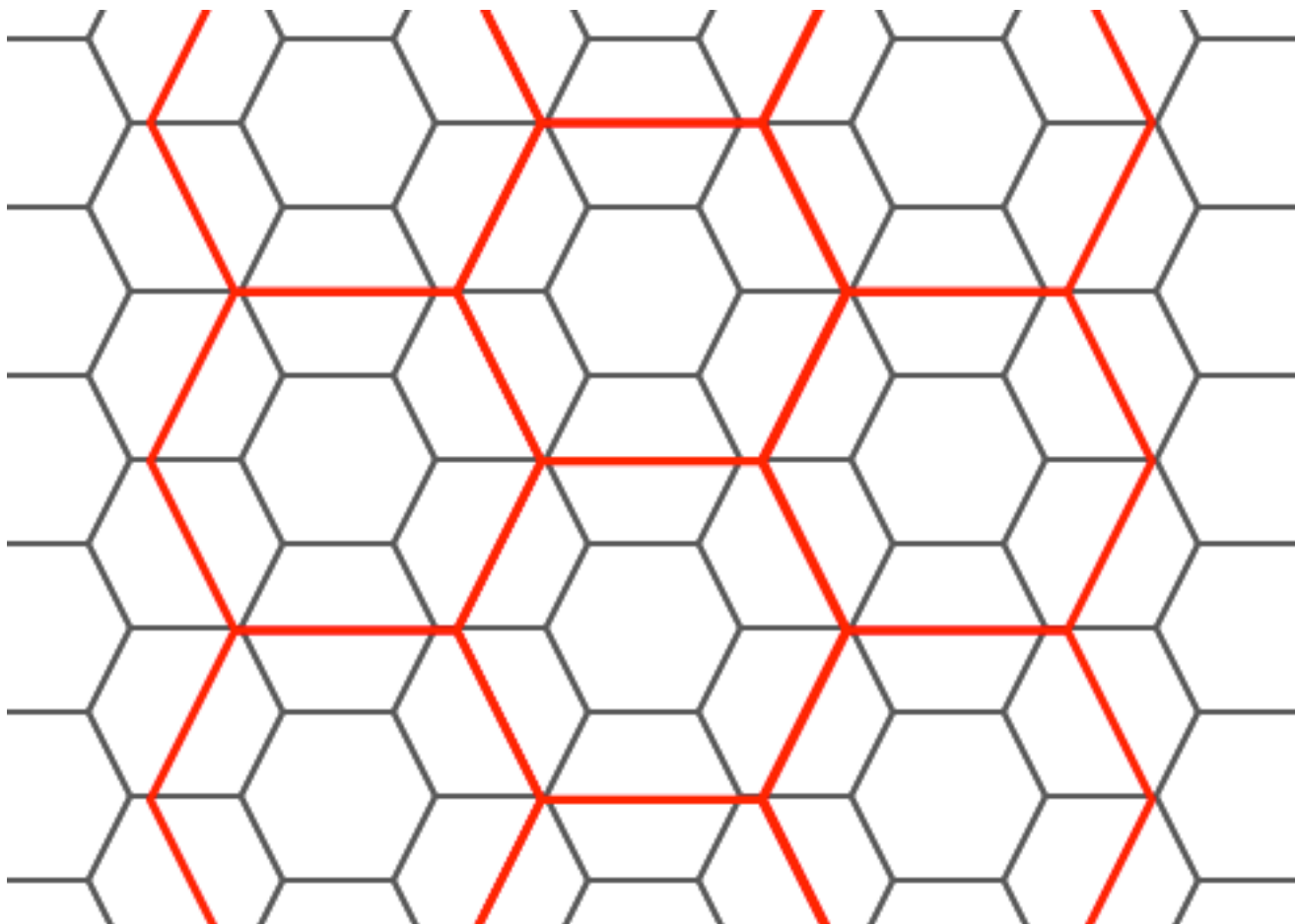
例

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



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

Doubling or tripling the edge size of the hexagon generates a new parent tiling that fits with the origin tiling. Unfortunately, it is not possible to generate parent hexagon tilings that the child tiles perfectly fit inside.



2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.

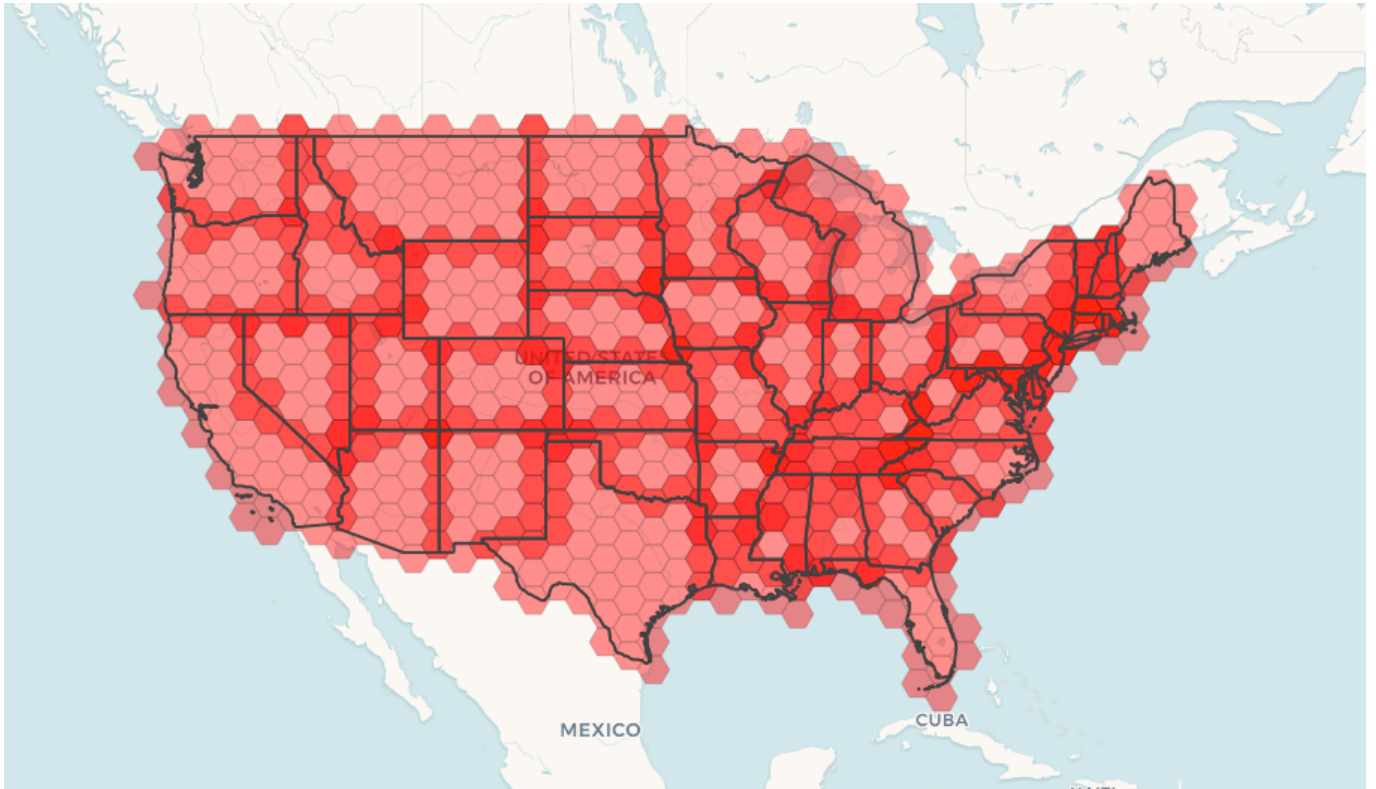
[illegible]

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;
```

:

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);

图例

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 文档.

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))
```

图例

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);

图例

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 文档.

例 7.3.16: 生成覆盖加拿大的正方形网格

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
```

例 7.3.17: 对点进行正方形网格摘要

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
```

例 7.3.18: 对点进行正方形网格摘要

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
```

例 7.3.19:

[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)');

图例

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 函数参考函数参考函数参考.

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))
```

图例

[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);

图例

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'), 1);
```



Letters generated by ST_Letters

Example: Scaling and moving words

```
SELECT ST_Translate(ST_Scale(ST_Letters('Yo'), 10, 10), 100,100);
```

例:

[ST_AsTWKB](#), [ST_Scale](#), [ST_Translate](#)

7.4 几何类型 (accessor)

7.4.1 GeometryType

GeometryType — ST_Geometry 几何类型的名称。

Synopsis

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

例:

ST_GeometryType(ST_GeomFromText('LINESTRING(0 0,1 1)', 4326)). 例: 'LINESTRING', 'POLYGON', 'MULTIPOINT' 等。

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

**Note**

PostGIS 3.6.1 中的 'POINTM' 数据类型支持 3D 点。

PostGIS 3.6.1 支持 3D 点，支持 TIN 数据类型。

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

SQL

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

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

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


ST_GeometryType

7.4.2 ST_Boundary

ST_Boundary — □□□□□□□□□□□□□□□□□□□□□□.

Synopsis

```
geometry ST_Boundary(geometry geomA);
```

[illegible]

GEOS ☒ ☒ ☒ ☒ ☒



Note

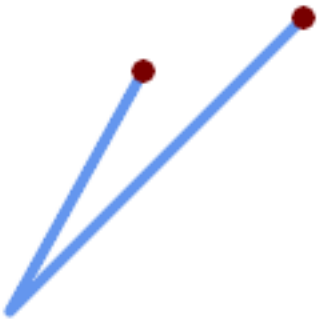
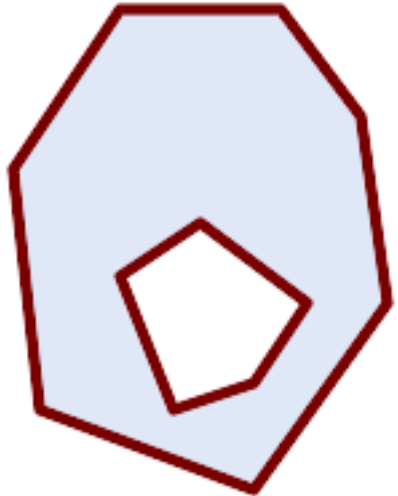
2.0.0 **GEOMETRYCOLLECTION**. 2.0.0
() NULL.

- ✔ This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**. OGC SPEC s2.1.1.1
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.17
- ✔ This function supports 3d and will not drop the z-index.

□□□□: 2.1.0 □□□□□□□□□□□□□□□□□□□□.

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



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

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

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

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

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

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

☐☐

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

7.4.3 ST_BoundingDiagonal

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

Synopsis

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

☐☐

返回给定几何图形的最小外接矩形的对角线。如果 `fits` 参数为 `true`，则返回的对角线将穿过几何图形的质心。如果 `fits` 参数为 `false`，则返回的对角线将穿过矩形的中心。如果几何图形是一个点，则返回该点的坐标。

`fits` 参数为 `true` (best fit) 时，返回的对角线将穿过几何图形的质心。如果 `fits` 参数为 `false`，则返回的对角线将穿过矩形的中心。如果几何图形是一个点，则返回该点的坐标。

返回的几何图形将具有与输入几何图形相同的 SRID。



Note

如果输入几何图形是一个点，则返回的对角线将是一个点。如果输入几何图形是一个线，则返回的对角线将是一个线。如果输入几何图形是一个面，则返回的对角线将是一个面。

2.2.0 版本引入。



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



This function supports M coordinates.

☐☐

```
-- Get the minimum X in a buffer around a point
SELECT ST_X(ST_StartPoint(ST_BoundingDiagonal(
  ST_Buffer(ST_Point(0,0),10)
)));
st_x
-----
-10
```

☐☐

[ST_StartPoint](#), [ST_EndPoint](#), [ST_X](#), [ST_Y](#), [ST_Z](#), [ST_M](#), &&&

7.4.4 ST_CoordDim

ST_CoordDim — ST_Geometry 函数。

Synopsis

integer **ST_CoordDim**(geometry geomA);

返回

ST_Geometry 函数。

MM 函数, **ST_NDims** 函数。

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

返回

```
SELECT ST_CoordDim('CIRCULARSTRING(1 2 3, 1 3 4, 5 6 7, 8 9 10, 11 12 13)');
      ---result--
      3

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

返回

ST_NDims

7.4.5 ST_Dimension

ST_Dimension — ST_Geometry 函数。

Synopsis

integer **ST_Dimension**(geometry g);

11

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 8



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

2.0.0 (polyhedral surface) TIN.



Note

[illegible]

This function supports Polyhedral surfaces.



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


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



ST NDims

7.4.6 ST Dump

ST Dump — Returns a set of geometry dump rows for the components of a geometry.

Synopsis

```
geometry dump[] ST Dump(geometry g1);
```



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

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

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

Version: 2.0.0, TIN

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



Note
1.3.4 版本中，`ST_Dump` 函数支持 `curve` 类型。1.3.4 版本中，`ST_Dump` 函数支持 `curve` 类型。

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

示例

```
SELECT sometable.field1, sometable.field1,
       (ST_Dump(sometable.geom)).geom AS geom
FROM sometable;

-- Break a compound curve into its constituent linestrings and circularstrings
SELECT ST_AsEWKT(a.geom), ST_HasArc(a.geom)
FROM ( SELECT (ST_Dump(p_geom)).geom AS geom
      FROM (SELECT ST_GeomFromEWKT('COMPOUNDCURVE(CIRCULARSTRING(0 0, 1 1, 1 0),(1 0, 0 1))') AS p_geom) AS b
      ) AS a;
      st_asewkt      | st_hasarc
-----+-----
CIRCULARSTRING(0 0,1 1,1 0) | t
LINESTRING(1 0,0 1)      | f
(2 rows)
```

示例，TIN 示例

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

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

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

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

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


geometry dump, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries

7.4.7 ST_DumpPoints

ST DumpPoints — .

Synopsis

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


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

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

To obtain a single geometry containing the coordinates use **ST Points**.

Enhanced: 2.1.0 Faster speed. Reimplemented as native-C.

Version: 2.0.0, TIN

1.5.0 ☒☒☒☒☒☒☒☒☒☒☒.

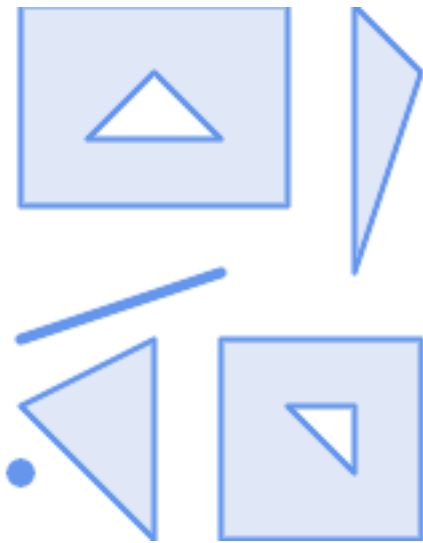
- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.

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

Classic Explode a Table of LineStrings into nodes

```
SELECT edge_id, (dp).path[1] As index, ST_AsText((dp).geom) As wktnode
FROM (SELECT 1 As edge_id
      , ST_DumpPoints(ST_GeomFromText('LINESTRING(1 2, 3 4, 10 10)')) AS dp
      UNION ALL
      SELECT 2 As edge_id
      , ST_DumpPoints(ST_GeomFromText('LINESTRING(3 5, 5 6, 9 10)')) AS dp
      ) As foo;
edge_id | index | wktnode
-----+-----+-----
1 | 1 | POINT(1 2)
1 | 2 | POINT(3 4)
1 | 3 | POINT(10 10)
2 | 1 | POINT(3 5)
2 | 2 | POINT(5 6)
2 | 3 | POINT(9 10)
```

图例



```
SELECT path, ST_AsText(geom)
FROM (
  SELECT (ST_DumpPoints(g.geom)).*
  FROM
    (SELECT
      'GEOMETRYCOLLECTION(
        POINT ( 0 1 ),
        LINESTRING ( 0 3, 3 4 ),
        POLYGON (( 2 0, 2 3, 0 2, 2 0 )),
        POLYGON (( 3 0, 3 3, 6 3, 6 0, 3 0 ),
          ( 5 1, 4 2, 5 2, 5 1 )),
        MULTIPOLYGON (
          (( 0 5, 0 8, 4 8, 4 5, 0 5 ),
            ( 1 6, 3 6, 2 7, 1 6 )),
```



```
        (( 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

```
-- Polyhedral surface cube --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
  (SELECT
    ST_DumpPoints(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 ←
    0)),
    ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
    ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
    ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ) AS gdump
  ) AS g;
-- result --
path | wkt
-----+-----
{1,1,1} | POINT(0 0 0)
{1,1,2} | POINT(0 0 1)
{1,1,3} | POINT(0 1 1)
{1,1,4} | POINT(0 1 0)
{1,1,5} | POINT(0 0 0)
{2,1,1} | POINT(0 0 0)
```

```

{2,1,2} | POINT(0 1 0)
{2,1,3} | POINT(1 1 0)
{2,1,4} | POINT(1 0 0)
{2,1,5} | POINT(0 0 0)
{3,1,1} | POINT(0 0 0)
{3,1,2} | POINT(1 0 0)
{3,1,3} | POINT(1 0 1)
{3,1,4} | POINT(0 0 1)
{3,1,5} | POINT(0 0 0)
{4,1,1} | POINT(1 1 0)
{4,1,2} | POINT(1 1 1)
{4,1,3} | POINT(1 0 1)
{4,1,4} | POINT(1 0 0)
{4,1,5} | POINT(1 1 0)
{5,1,1} | POINT(0 1 0)
{5,1,2} | POINT(0 1 1)
{5,1,3} | POINT(1 1 1)
{5,1,4} | POINT(1 1 0)
{5,1,5} | POINT(0 1 0)
{6,1,1} | POINT(0 0 1)
{6,1,2} | POINT(1 0 1)
{6,1,3} | POINT(1 1 1)
{6,1,4} | POINT(0 1 1)
{6,1,5} | POINT(0 0 1)
(30 rows)

```

```

-- Triangle --
SELECT (g.gdump).path, ST_AsText((g.gdump).geom) as wkt
FROM
  (SELECT
    ST_DumpPoints( ST_GeomFromEWKT('TRIANGLE ((
      0 0,
      0 9,
      9 0,
      0 0
    ))') ) AS gdump
  ) AS g;
-- result --
path |      wkt
-----+-----
{1}  | POINT(0 0)
{2}  | POINT(0 9)
{3}  | POINT(9 0)
{4}  | POINT(0 0)

```

```

-- TIN --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
  (SELECT
    ST_DumpPoints( ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))
  )') ) AS gdump

```

```
) AS g;
-- result --
path | wkt
-----+-----
{1,1,1} | POINT(0 0 0)
{1,1,2} | POINT(0 0 1)
{1,1,3} | POINT(0 1 0)
{1,1,4} | POINT(0 0 0)
{2,1,1} | POINT(0 0 0)
{2,1,2} | POINT(0 1 0)
{2,1,3} | POINT(1 1 0)
{2,1,4} | POINT(0 0 0)
(8 rows)
```

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

7.4.8 ST_DumpSegments

`ST_DumpSegments` —

Synopsis

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

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

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

Availability: 3.2.0

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

```
SELECT path, ST_AsText(geom)
FROM (
  SELECT (ST_DumpSegments(g.geom)).*
  FROM (SELECT 'GEOMETRYCOLLECTION(
    LINESTRING(1 1, 3 3, 4 4),
    POLYGON((5 5, 6 6, 7 7, 5 5))
  )'::geometry AS geom
        ) AS g
) j;
```

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

(5 rows)

几何类型, **TIN** 几何类型

```
-- Triangle --
SELECT path, ST_AsText(geom)
FROM (
  SELECT (ST_DumpSegments(g.geom)).*
  FROM (SELECT 'TRIANGLE((
    0 0,
    0 9,
    9 0,
    0 0
  ))'::geometry AS geom
        ) AS g
) j;
```

path	b'' b''	st_astext
{1,1}	b'' b''	LINESTRING(0 0,0 9)
{1,2}	b'' b''	LINESTRING(0 9,9 0)
{1,3}	b'' b''	LINESTRING(9 0,0 0)

(3 rows)

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

) j;

path	b''	b''	st_asewkt
{1,1,1}	b''	b''	LINESTRING(0 0 0,0 0 1)
{1,1,2}	b''	b''	LINESTRING(0 0 1,0 1 0)
{1,1,3}	b''	b''	LINESTRING(0 1 0,0 0 0)
{2,1,1}	b''	b''	LINESTRING(0 0 0,0 1 0)
{2,1,2}	b''	b''	LINESTRING(0 1 0,1 1 0)
{2,1,3}	b''	b''	LINESTRING(1 1 0,0 0 0)

(6 rows)

geometry dump, ST Collect, ST Dump, ST NumInteriorRing,

7.4.9 ST_DumpRings

ST_DumpRings — Returns a set of `geometry_dump` rows for the exterior and interior rings of a Polygon.

Synopsis

```
geometry_dump[] ST_DumpRings(geometry a_polygon);
```


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

The *geom* field contains each ring as a POLYGON. The *path* field is an integer array of length 1 containing the polygon ring index. The exterior ring (shell) has index 0. The interior rings (holes) have indices of 1 and higher.



Note

XXXXXXXXXXXXXXXXXXXXX. XXXXXX ST Dump XXXXXXXXXXXXXXX.

Availability: PostGIS 1.1.3. Requires PostgreSQL 7.3 or higher.



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



General form of query.

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

A polygon with a single hole.

图 11.1

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

SELECT ST_AsText(ST_Envelope('LINESTRING(0 0, 1 3)::geometry'));
      st_astext
-----
POLYGON((0 0,0 3,1 3,1 0,0 0))
(1 row)

SELECT ST_AsText(ST_Envelope('POLYGON((0 0, 0 1, 1.0000001 1, 1.0000001 0, 0 0))::geometry' ←
));
      st_astext
-----
POLYGON((0 0,0 1,1.00000011920929 1,1.00000011920929 0,0 0))
(1 row)
SELECT ST_AsText(ST_Envelope('POLYGON((0 0, 0 1, 1.0000000001 1, 1.0000000001 0, 0 0))::' ←
geometry));
      st_astext
-----
POLYGON((0 0,0 1,1.00000011920929 1,1.00000011920929 0,0 0))
(1 row)

SELECT Box3D(geom), Box2D(geom), ST_AsText(ST_Envelope(geom)) As envelopewkt
FROM (SELECT 'POLYGON((0 0, 0 1000012333334.34545678, 1.0000001 1, 1.0000001 0, 0 0' ←
0))::geometry As geom) As foo;
```



Envelope of a point and linestring.

```
SELECT ST_AsText(ST_Envelope(
      ST_Collect(
        ST_GeomFromText('LINESTRING(55 75,125 150)'),
        ST_Point(20, 80))
```



```

                                )) As wktenv;
wktenv
-----
POLYGON((20 75,20 150,125 150,125 75,20 75))

```

☐☐

Box2D, Box3D, ST_OrientedEnvelope

7.4.12 ST_ExteriorRing

ST_ExteriorRing — 返回多边形的外环。

Synopsis

geometry **ST_ExteriorRing**(geometry a_polygon);

☐☐

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



Note

ST_ExteriorRing 返回的是多边形的外环。ST_Dump 返回的是所有环。

- ✔ This method implements the **OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1**
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 8.2.3, 8.3.3
- ✔ This function supports 3d and will not drop the z-index.

☐☐

```

--If you have a table of polygons
SELECT gid, ST_ExteriorRing(geom) AS ering
FROM sometable;

--If you have a table of MULTIPOLYGONS
--and want to return a MULTILINESTRING composed of the exterior rings of each polygon
SELECT gid, ST_Collect(ST_ExteriorRing(geom)) AS erings
      FROM (SELECT gid, (ST_Dump(geom)).geom As geom
            FROM sometable) As foo
GROUP BY gid;

--3d Example
SELECT ST_AsEWKT(
  ST_ExteriorRing(
    ST_GeomFromEWKT('POLYGON((0 0 1, 1 1 1, 1 2 1, 1 1 1, 0 0 1))')
  )
)

```

```

    )
);

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

```

函数

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

7.4.13 ST_GeometryN

ST_GeometryN — ST_Geometry 子几何体。

Synopsis

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

函数

返回几何体 geomA 的第 n 个子几何体。如果 geomA 是多曲线 (multicurve) 则返回第 n 个子曲线。n 的范围是 1 到 N，如果 n 超出范围则返回 NULL。



Note

0.8.0 版本之前 OGC 规范是 1-索引的。从 0-索引开始。



Note

ST_Dump 函数返回子几何体，ST_GeometryN 返回子几何体。

版本: 2.0.0 版本之前，返回 TIN 子几何体。

版本: 2.0.0 版本之前，返回 NULL。2.0.0 版本 ST_GeometryN(..,1) 返回子几何体。

- ✓ 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).

实验四

```
--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);
```

实验四, TIN 实验四

```
-- Polyhedral surface example
-- Break a Polyhedral surface into its faces
SELECT ST_AsEWKT(ST_GeometryN(p_geom,3)) As geom_ewkt
FROM (SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE(
((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)') AS p_geom ) AS a;
```

geom_ewkt
POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0))

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



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

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

☒☒

GeometryType

7.4.15 ST_HasArc

ST_HasArc — Tests if a geometry contains a circular arc

Synopsis

boolean **ST_HasArc**(geometry geomA);

☒☒

☒☒☒☒☒☒☒☒☒, ☒☒☒, ☒☒☒☒☒☒☒ TRUE ☒☒☒☒☒☒.

1.2.2 ☒☒☒☒☒☒☒☒☒☒☒.



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



This method supports Circular Strings and Curves.

例

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

例

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

7.4.16 ST_InteriorRingN

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

Synopsis

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

例

返回第 `N` 个内部环。如果 `N` 为负数，则返回第 `N` 个外部环 (range) 的 NULL 值。



Note

如果 `N` 为负数，则返回第 `N` 个外部环。使用 `ST_Dump` 可以更容易地访问内部环。

- ✓ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✓ This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5
- ✓ This function supports 3d and will not drop the z-index.

例

```
SELECT ST_AsText(ST_InteriorRingN(geom, 1)) As geom
FROM (SELECT ST_BuildArea(
      ST_Collect(ST_Buffer(ST_Point(1,2), 20,3),
      ST_Buffer(ST_Point(1, 2), 10,3))) As geom
      ) as foo;
```

例

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

7.4.17 ST_NumCurves

ST_NumCurves — Return the number of component curves in a `CompoundCurve`.

Synopsis

integer **ST_NumCurves**(geometry a_compoundcurve);

⊠

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



This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5



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

⊠

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

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

⊠

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

7.4.18 ST_CurveN

ST_CurveN — Returns the Nth component curve geometry of a `CompoundCurve`.

Synopsis

geometry **ST_CurveN**(geometry a_compoundcurve, integer index);

⊠

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



This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5



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

例

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

例

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

7.4.19 ST_IsClosed

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

Synopsis

boolean **ST_IsClosed**(geometry g);

例

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

- ✓ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✓ This method implements the SQL/MM specification. SQL-MM 3: 7.1.5, 9.3.3



Note

SQL-MM defines the result of **ST_IsClosed**(NULL) to be 0, while PostGIS returns NULL.

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

例


```

postgis=# SELECT ST_IsClosed('LINESTRING(0 0, 1 1)::geometry');
st_isclosed
-----
f
(1 row)

postgis=# SELECT ST_IsClosed('LINESTRING(0 0, 0 1, 1 1, 0 0)::geometry');
st_isclosed
-----
t
(1 row)

postgis=# SELECT ST_IsClosed('MULTILINESTRING((0 0, 0 1, 1 1, 0 0),(0 0, 1 1))::geometry');
st_isclosed
-----
f
(1 row)

postgis=# SELECT ST_IsClosed('POINT(0 0)::geometry');
st_isclosed
-----
t
(1 row)

postgis=# SELECT ST_IsClosed('MULTIPOINT((0 0), (1 1))::geometry');
st_isclosed
-----
t
(1 row)

```

测试测试测试测试

```

-- A cube --
SELECT ST_IsClosed(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)
),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)
) )')));

st_isclosed
-----
t

-- Same as cube but missing a side --
SELECT ST_IsClosed(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0
0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)
),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)) )')));

st_isclosed
-----
f

```

函数

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);

函数

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.

函数

```
postgis=# SELECT ST_IsCollection('LINestring(0 0, 1 1)::geometry');
st_iscollection
-----
f
(1 row)

postgis=# SELECT ST_IsCollection('MULTIPOINT EMPTY)::geometry');
st_iscollection
-----
t
(1 row)

postgis=# SELECT ST_IsCollection('MULTIPOINT((0 0))::geometry');
st_iscollection
-----
t
(1 row)

postgis=# SELECT ST_IsCollection('MULTIPOINT((0 0), (42 42))::geometry');
st_iscollection
```

```

-----
t
(1 row)

postgis=# SELECT ST_IsCollection('GEOMETRYCOLLECTION(POINT(0 0))'::geometry);
st_iscollection
-----
t
(1 row)

```

☐☐

ST_NumGeometries

7.4.21 ST_IsEmpty

ST_IsEmpty — Tests if a geometry is empty.

Synopsis

boolean **ST_IsEmpty**(geometry geomA);

☐☐

☐☐☐☐☐☐☐☐☐☐ TRUE ☐☐☐☐☐☐. TRUE ☐☐☐, ☐☐☐☐☐☐☐☐☐☐, ☐☐☐, ☐☐☐☐☐☐☐☐☐☐☐☐.



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 ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐.

例

```
SELECT ST_IsEmpty(ST_GeomFromText('GEOMETRYCOLLECTION EMPTY'));
st_isempty
-----
t
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON EMPTY'));
st_isempty
-----
t
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))'));
st_isempty
-----
f
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2)))') = false;
?column?
-----
t
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('CIRCULARSTRING EMPTY'));
st_isempty
-----
t
(1 row)
```

7.4.22 ST_IsPolygonCCW

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

Synopsis

boolean **ST_IsPolygonCCW** (geometry geom);

例

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

Returns true if the geometry has no polygonal components.



Note

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

**Note**

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

2.2.0 简体中文。



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



This function supports M coordinates.

☒

`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 简体中文。



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



This function supports M coordinates.

Synopsis

boolean **ST_IsSimple**(geometry geomA);

返回

如果几何体是简单的，则返回 TRUE。如果几何体不是简单的，则返回 FALSE。OGC 简单特征实现规范 (OGC Simple Features Implementation Specification) 中，"OpenGIS 简单特征实现规范 (Ensuring OpenGIS compliance of geometries)" 部分。



Note

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

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1](#)
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.8
- ✔ This function supports 3d and will not drop the z-index.

返回

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

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

返回

ST_IsValid

7.4.26 ST_M

ST_M — Returns the M coordinate of a Point.

Synopsis

float **ST_M**(geometry a_point);

返回

返回 `M` 几何体。 `M` 几何体 `NULL` 返回 `NULL`。 返回类型 `geometry`。



Note

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

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

返回

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

返回

[ST_GeomFromEWKT](#), [ST_X](#), [ST_Y](#), [ST_Z](#)

7.4.27 ST_MemSize

`ST_MemSize` — `ST_Geometry` 内存大小。

Synopsis

integer **ST_MemSize**(geometry geomA);

返回

`ST_Geometry` 内存大小。

This complements the PostgreSQL built-in [database object functions](#) `pg_column_size`, `pg_size_pretty`, `pg_relation_size`, `pg_total_relation_size`.

Note



`pg_relation_size` which gives the byte size of a table may return byte size lower than `ST_MemSize`. This is because `pg_relation_size` does not add toasted table contribution and large geometries are stored in TOAST tables.

`pg_total_relation_size` 返回, TOAST 返回。

`pg_column_size` returns how much space a geometry would take in a column considering compression, so may be lower than `ST_MemSize`

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

Changed: 2.2.0 name changed to ST_MemSize to follow naming convention.

☒

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

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

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

```
73
```

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

7.4.28 ST_NDims

ST_NDims — ST_Geometry 数据类型函数。

Synopsis

integer **ST_NDims**(geometry g1);

☒

PostGIS 2 - 2 (x,y), 3 - 3 (x,y,z), 3 - 2 (x,y,m), 4 - 3 (x,y,z,m) 数据类型。

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

例

```
SELECT ST_NDims(ST_GeomFromText('POINT(1 1)')) As d2point,
       ST_NDims(ST_GeomFromEWKT('POINT(1 1 2)')) As d3point,
       ST_NDims(ST_GeomFromEWKT('POINTM(1 1 0.5)')) As d2pointm;

 d2point | d3point | d2pointm
-----+-----+-----
      2 |      3 |      3
```

例

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

7.4.29 ST_NPoints

ST_NPoints — 返回几何体中的点数量。

Synopsis

integer **ST_NPoints**(geometry g1);

例

返回几何体中的点数量。对于多面体表面，返回其顶点的数量。
 2.0.0 版本开始支持 (polyhedral surface) 几何体。



Note

1.3.4 版本开始支持 (curve) 几何体。1.3.4 版本开始支持 (polyhedral surface) 几何体。

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

例

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

--Polygon in 3D space
SELECT ST_NPoints(ST_GeomFromEWKT('LINESTRING(77.29 29.07 1,77.42 29.26 0,77.27 29.31 -1,77.29 29.07 3)'));
--result
4
```

返回

ST_NumPoints

7.4.30 ST_NRings

ST_NRings — 返回几何体中的环数。

Synopsis

integer **ST_NRings**(geometry geomA);

返回

NumInteriorRings 返回几何体中的内部环数。NumInteriorRings 返回几何体中的内部环数。

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

返回

```
SELECT ST_NRings(geom) As Nrings, ST_NumInteriorRings(geom) As ninterrings
FROM (SELECT ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))') As geom) As foo;
```

	nrings		ninterrings
(1 row)	1		0

返回

ST_NumInteriorRings

7.4.31 ST_NumGeometries

ST_NumGeometries — 返回几何体中的几何体数量。

Synopsis

integer **ST_NumGeometries**(geometry geom);

ST_NumGeometries

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

更新: 2.0.0 支持多部件几何体, 支持 TIN 几何体。

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

- ✔ This method implements the SQL/MM specification. SQL-MM 3: 9.1.4
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

示例

```
--Prior versions would have returned NULL for this -- in 2.0.0 this returns 1
SELECT ST_NumGeometries(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
--result
1

--Geometry Collection Example - multis count as one geom in a collection
SELECT ST_NumGeometries(ST_GeomFromEWKT('GEOMETRYCOLLECTION(MULTIPOINT((-2 3),(-2 2)),
LINESTRING(5 5 ,10 10),
POLYGON((-7 4.2,-7.1 5,-7.1 4.3,-7 4.2)))'));
--result
3
```

函数

ST_GeometryN, **ST_Multi**

7.4.32 ST_NumInteriorRings

ST_NumInteriorRings — 返回多边形内部环的数量。

Synopsis

integer **ST_NumInteriorRings**(geometry a_polygon);

函数

更新: 2.0.0 支持多部件几何体。空几何体返回 NULL。

- ✔ This method implements the SQL/MM specification. SQL-MM 3: 8.2.5
- 更新: 2.0.0 支持多部件几何体。

例

```
--If you have a regular polygon
SELECT gid, field1, field2, ST_NumInteriorRings(geom) AS numholes
FROM sometable;

--If you have multipolygons
--And you want to know the total number of interior rings in the MULTIPOLYGON
SELECT gid, field1, field2, SUM(ST_NumInteriorRings(geom)) AS numholes
FROM (SELECT gid, field1, field2, (ST_Dump(geom)).geom As geom
      FROM sometable) As foo
GROUP BY gid, field1,field2;
```

例

[ST_NumInteriorRing](#), [ST_PointN](#)

7.4.33 ST_NumInteriorRing

`ST_NumInteriorRing` — 返回多边形中内部环的数量。 `ST_NumInteriorRings` 的别名。

Synopsis

integer **ST_NumInteriorRing**(geometry a_polygon);

例

[ST_NumInteriorRings](#), [ST_PointN](#)

7.4.34 ST_NumPatches

`ST_NumPatches` — 返回几何体中补丁的数量。如果几何体是 NULL，则返回 NULL。

Synopsis

integer **ST_NumPatches**(geometry g1);

例

返回几何体中补丁的数量。如果几何体是 NULL，则返回 NULL。 `ST_NumGeometries` 的别名。 `MM` 返回几何体中补丁的数量。 `ST_NumGeometries` 的别名。

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.

例

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

例

[ST_GeomFromEWKT](#), [ST_NumGeometries](#)

7.4.35 ST_NumPoints

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

Synopsis

integer **ST_NumPoints**(geometry g1);

例

`ST_LineString` 或 `ST_CircularString` 的几何体中的点数量。 1.4 版本中，`ST_NumPoints` 返回的点是几何体中的点，而不是几何体中的段。 1.4 版本中，`ST_NumPoints` 返回的点是几何体中的点，而不是几何体中的段。 `ST_NPoints` 返回的点是几何体中的点，而不是几何体中的段。 `ST_NPoints` 返回的点是几何体中的点，而不是几何体中的段。

- ✓ 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

例

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

例

[ST_NPoints](#)

7.4.36 ST_PatchN

ST_PatchN — ST_Geometry 函数。

Synopsis

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

返回

如果 geomA 是 POLYHEDRALSURFACE, POLYHEDRALSURFACEM 或 1-N 面体 (面) 的集合, 则返回第 n 个面。如果 geomA 是 NULL, 则返回 NULL。如果 n 不在 1 到 N 的范围内, 则返回 ST_GeometryN 的结果。ST_GeometryN 返回第 n 个面。



Note

如果 n 不在 1 到 N 的范围内, 则返回 NULL。



Note

如果 geomA 是 POLYHEDRALSURFACE, 则 ST_Dump 返回每个面的几何体。

2.0.0 版本引入。

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

返回

```
--Extract the 2nd face of the polyhedral surface
SELECT ST_AsEWKT(ST_PatchN(geom, 2)) As geomewkt
FROM (
VALUES (ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ) )
As foo(geom);

geomewkt
-----+-----
POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))
```

返回

[ST_AsEWKT](#), [ST_GeomFromEWKT](#), [ST_Dump](#), [ST_GeometryN](#), [ST_NumGeometries](#)

7.4.37 ST_PointN

ST_PointN — ST_LineString 和 ST_CircularString 的变体函数。

Synopsis

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

返回

返回第 N 个点。如果 N 为负数，则返回第 -N 个点。如果 N 为 0，则返回 NULL。



Note

0.8.0 版本实现了 OGC 的 1-版本。OGC 的 2-版本 (2.0.0) 已经实现。0-版本尚未实现。



Note

返回第 N 个点，如果 N 为负数，则返回第 -N 个点。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: 7.2.5, 7.3.5
- ✓ This function supports 3d and will not drop the z-index.
- ✓ This method supports Circular Strings and Curves.



Note

2.0.0 版本实现了 OGC 的 2-版本。PostGIS 的 2.0.0 版本已经实现。2.0.0 版本尚未实现。NULL 值。2.3.0 版本实现了 (-1 版本) 的变体。

返回

```
-- Extract all POINTs from a LINESTRING
SELECT ST_AsText(
  ST_PointN(
    column1,
    generate_series(1, ST_NPoints(column1))
  ))
FROM ( VALUES ('LINESTRING(0 0, 1 1, 2 2)::geometry') ) AS foo;

st_astext
-----
POINT(0 0)
```



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

--Example circular string
SELECT ST_AsText(ST_PointN(ST_GeomFromText('CIRCULARSTRING(1 2, 3 2, 1 2)'), 2));

    st_astext
-----
POINT(3 2)
(1 row)

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

    st_astext
-----
POINT Z (1 1 1)
(1 row)
```



ST_RemoveRepeatedPoints, ST_PointN

7.4.39 ST_StartPoint

ST_StartPoint — Returns the first point of a LineString.

Synopsis

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



LINESTRING ☒ ☒ CIRCULARLINESTRING ☒☒☒☒☒ POINT ☒☒☒☒☒ ☒☒☒☒☒
LINESTRING ☒ ☒ CIRCULARLINESTRING ☒☒☒☒ NULL ☒☒☒☒☒.

- ✔ This method implements the SQL/MM specification. SQL-MM 3: 7.1.3
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.

Note

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

PostGIS 2.0.0. 2.0.0. NULL. 2.0.0. NULL.

Start point of a LineString

```
SELECT ST_AsText(ST_StartPoint('LINESTRING(0 1, 0 2)::geometry'));
st_astext
-----
POINT(0 1)
```

Start point of a non-LineString is NULL

```
SELECT ST_StartPoint('POINT(0 1)::geometry') IS NULL AS is_null;
is_null
-----
t
```

Start point of a 3D LineString

```
SELECT ST_AsEWKT(ST_StartPoint('LINESTRING(0 1 1, 0 2 2)::geometry'));
st_asewkt
-----
POINT(0 1 1)
```

`ST_LineString` 和 `ST_CircularString` 函数支持。

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

和

`ST_EndPoint`, `ST_PointN`

7.4.40 ST_Summary

`ST_Summary` — 返回几何体的摘要字符串。

Synopsis

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

和

返回摘要字符串。
返回摘要字符串的格式如下：

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



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



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

1.2.2 函数支持。

版本: 2.0.0 函数支持。

版本: 2.1.0 和。返回摘要字符串的格式如下 S 值。

版本: 2.2.0 函数支持 TIN 几何体 (curve) 值。

例

```
=# SELECT ST_Summary(ST_GeomFromText('LINESTRING(0 0, 1 1)')) as geom,
         ST_Summary(ST_GeogFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) geog;
-----+-----
LineString[B] with 2 points | Polygon[BGS] with 1 rings
                           | ring 0 has 5 points
                           :
(1 row)

=# SELECT ST_Summary(ST_GeogFromText('LINESTRING(0 0 1, 1 1 1)')) As geog_line,
         ST_Summary(ST_GeomFromText('SRID=4326;POLYGON((0 0 1, 1 1 2, 1 2 3, 1 1 1, 0 0 1))' ←
         ')) As geom_poly;
;
-----+-----
geog_line | geom_poly
-----+-----
LineString[ZBGS] with 2 points | Polygon[ZBS] with 1 rings
                           | ring 0 has 5 points
                           :
(1 row)
```

例

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

7.4.41 ST_X

ST_X — Returns the X coordinate of a Point.

Synopsis

float **ST_X**(geometry a_point);

例

例 例 X 例 例 例 例 例. X 例 例 例 例 例 NULL 例 例 例 例. 例 例 例 例 例 例 例 例.



Note
To get the minimum and maximum X value of geometry coordinates use the functions [ST_XMin](#) and [ST_XMax](#).

- ✔ This method implements the SQL/MM specification. SQL-MM 3: 6.1.3
- ✔ This function supports 3d and will not drop the z-index.

例

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

SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
 st_y
-----
    1.5
(1 row)
```

例

[ST_Centroid](#), [ST_GeomFromEWKT](#), [ST_M](#), [ST_XMax](#), [ST_XMin](#), [ST_Y](#), [ST_Z](#)

7.4.42 ST_Y

ST_Y — Returns the Y coordinate of a Point.

Synopsis

float **ST_Y**(geometry a_point);

例

返回 Y 坐标。Y 坐标为 NULL 时返回 NULL。支持 3D 几何。



Note

To get the minimum and maximum Y value of geometry coordinates use the functions [ST_YMin](#) and [ST_YMax](#).

- ✓ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✓ This method implements the SQL/MM specification. SQL-MM 3: 6.1.4
- ✓ This function supports 3d and will not drop the z-index.

例

```
SELECT ST_Y(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_y
-----
      2
(1 row)
```

```
SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
st_y
-----
1.5
(1 row)
```

☐☐

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

7.4.43 ST_Z

ST_Z — Returns the Z coordinate of a Point.

Synopsis

float **ST_Z**(geometry a_point);

☐☐

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



Note

To get the minimum and maximum Z value of geometry coordinates use the functions [ST_ZMin](#) and [ST_ZMax](#).



This method implements the SQL/MM specification.



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

☐☐

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

☐☐

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

7.4.44 ST_Zmflag

ST_Zmflag — ST_Geometry 的 Z 标志。

Synopsis

smallint **ST_Zmflag**(geometry geomA);

返回

ST_Geometry 的 Z 标志。

Values are: 0 = 2D, 1 = 3D-M, 2 = 3D-Z, 3 = 4D.

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

返回

```
SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRING(1 2, 3 4)'));
st_zmflag
-----
0

SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRINGM(1 2 3, 3 4 3)'));
st_zmflag
-----
1

SELECT ST_Zmflag(ST_GeomFromEWKT('CIRCULARSTRING(1 2 3, 3 4 3, 5 6 3)'));
st_zmflag
-----
2

SELECT ST_Zmflag(ST_GeomFromEWKT('POINT(1 2 3 4)'));
st_zmflag
-----
3
```

返回

ST_CoordDim, ST_NDims, ST_Dimension

7.4.45 ST_HasZ

ST_HasZ — Checks if a geometry has a Z dimension.

Synopsis

boolean **ST_HasZ**(geometry geom);



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

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

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

Availability: 3.5.0



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



This function supports M coordinates.



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

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



ST_Zmflag

ST_HasM

7.4.46 ST_HasM

ST_HasM — Checks if a geometry has an M (measure) dimension.

Synopsis

boolean **ST_HasM**(geometry geom);



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

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

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

Availability: 3.5.0



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



This function supports M coordinates.

SQL

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

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

SQL

ST_Zmflag

ST_HasZ

7.5 几何函数 (editor)

7.5.1 ST_AddPoint

ST_AddPoint — 向 LineString 添加点。

Synopsis

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

SQL

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

1.1.0 引入。



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

SQL

Add a point to the end of a 3D line

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

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

Guarantee all lines in a table are closed by adding the start point of each line to the end of the line only for those that are not closed.

```
UPDATE sometable
SET geom = ST_AddPoint(geom, ST_StartPoint(geom))
FROM sometable
WHERE ST_IsClosed(geom) = false;
```

☒☒

[ST_RemovePoint](#), [ST_SetPoint](#)

7.5.2 ST_CollectionExtract

ST_CollectionExtract — Given a geometry collection, returns a multi-geometry containing only elements of a specified type.

Synopsis

geometry **ST_CollectionExtract**(geometry collection);

geometry **ST_CollectionExtract**(geometry collection, integer type);

☒☒

Given a geometry collection, returns a homogeneous multi-geometry.

If the *type* is not specified, returns a multi-geometry containing only geometries of the highest dimension. So polygons are preferred over lines, which are preferred over points.

If the *type* is specified, returns a multi-geometry containing only that type. If there are no sub-geometries of the right type, an EMPTY geometry is returned. Only points, lines and polygons are supported. The type numbers are:

- 1 == POINT
- 2 == LINESTRING
- 3 == POLYGON

For atomic geometry inputs, the geometry is returned unchanged if the input type matches the requested type. Otherwise, the result is an EMPTY geometry of the specified type. If required, these can be converted to multi-geometries using [ST_Multi](#).



Warning

MultiPolygon results are not checked for validity. If the polygon components are adjacent or overlapping the result will be invalid. (For example, this can occur when applying this function to an [ST_Split](#) result.) This situation can be checked with [ST_IsValid](#) and repaired with [ST_MakeValid](#).

1.5.0 ☒☒☒☒☒☒☒☒☒☒.



Note

Prior to 1.5.3 this function returned atomic inputs unchanged, no matter type. In 1.5.3 non-matching single geometries returned a NULL result. In 2.0.0 non-matching single geometries return an EMPTY result of the requested type.

例

Extract highest-dimension type:

```
SELECT ST_AsText(ST_CollectionExtract(
    'GEOMETRYCOLLECTION( POINT(0 0), LINESTRING(1 1, 2 2) )'));
st_astext
-----
MULTILINESTRING((1 1, 2 2))
```

Extract points (type 1 == POINT):

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

Extract lines (type 2 == LINESTRING):

```
SELECT ST_AsText(ST_CollectionExtract(
    'GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(LINESTRING(0 0, 1 1)),LINESTRING(2 2, 3 3))' ↔
    ,
    2 ));
st_astext
-----
MULTILINESTRING((0 0, 1 1), (2 2, 3 3))
```

例

`ST_CollectionHomogenize`, `ST_Multi`, `ST_IsValid`, `ST_MakeValid`

7.5.3 ST_CollectionHomogenize

`ST_CollectionHomogenize` — Returns the simplest representation of a geometry collection.

Synopsis

geometry **`ST_CollectionHomogenize`**(geometry collection);

例

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

- Homogeneous (uniform) collections are returned as the appropriate multi-geometry.
- Heterogeneous (mixed) collections are flattened into a single GeometryCollection.
- Collections containing a single atomic element are returned as that element.
- Atomic geometries are returned unchanged. If required, these can be converted to a multi-geometry using **`ST_Multi`**.



Warning

This function does not ensure that the result is valid. In particular, a collection containing adjacent or overlapping Polygons will create an invalid MultiPolygon. This situation can be checked with `ST_IsValid` and repaired with `ST_MakeValid`.

2.0.0 □□□□□□□□□□.



Single-element collection converted to an atomic geometry

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

Nested single-element collection converted to an atomic geometry:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(MULTIPOINT((0 0)))'));
      st_astext
-----
POINT(0 0)
```

Collection converted to a multi-geometry:

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

Nested heterogeneous collection flattened to a GeometryCollection:

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

st_astext
GEOMETRYCOLLECTION(POINT(0 0),LINESTRING(1 1,2 2))

Collection of Polygons converted to an (invalid) MultiPolygon:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION (POLYGON ((10 50, 50 50, 50 10, 10 10, 10 50)), POLYGON ((90 50, 90 10, 50 10, 50 50, 90 50)))'));

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



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);

国国

Converts a CIRCULAR STRING to regular LINESTRING or CURVEPOLYGON to POLYGON or MULTISURFACE to MULTIPOLYGON. Useful for outputting to devices that can't support CIRCULARSTRING geometry types

Converts a given geometry to a linear geometry. Each curved geometry or segment is converted into a linear approximation using the given 'tolerance' and options (32 segments per quadrant and no options by default).

The 'tolerance_type' argument determines interpretation of the 'tolerance' argument. It can take the following values:

- 0 (default): Tolerance is max segments per quadrant.
- 1: Tolerance is max-deviation of line from curve, in source units.
- 2: Tolerance is max-angle, in radians, between generating radii.

The 'flags' argument is a bitfield. 0 by default. Supported bits are:

- 1: Symmetric (orientation independent) output.
- 2: Retain angle, avoids reducing angles (segment lengths) when producing symmetric output. Has no effect when Symmetric flag is off.

Availability: 1.3.0

Enhanced: 2.4.0 added support for max-deviation and max-angle tolerance, and for symmetric output.

Enhanced: 3.0.0 implemented a minimum number of segments per linearized arc to prevent topological collapse.

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
 - ✔ This method implements the SQL/MM specification. SQL-MM 3: 7.1.7
 - ✔ This function supports 3d and will not drop the z-index.
 - ✔ This method supports Circular Strings and Curves.
-

❏

```
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)')));

--Result --
LINESTRING(220268 150415,220269.95064912 150416.539364228,220271.823415575 150418.17258804,220273.613787707 150419.895736857,
220275.317452352 150421.704659462,220276.930305234 150423.594998003,220278.448460847 150425.562198489,
220279.868261823 150427.60152176,220281.186287736 150429.708054909,220282.399363347 150431.876723113,
220283.50456625 150434.10230186,220284.499233914 150436.379429536,220285.380970099 150438.702620341,220286.147650624 150441.066277505,
220286.797428488 150443.464706771,220287.328738321 150445.892130112,220287.740300149 150448.342699654,
220288.031122486 150450.810511759,220288.200504713 150453.289621251,220288.248038775 150455.77405574,
220288.173610157 150458.257830005,220287.977398166 150460.734960415,220287.659875492 150463.199479347,
220287.221807076 150465.64544956,220286.664248262 150468.066978495,220285.988542259 150470.458232479,220285.196316903 150472.81345077,
220284.289480732 150475.126959442,220283.270218395 150477.39318505,220282.140985384 150479.606668057,
220280.90450212 150481.762075989,220279.5637474 150483.85421628,220278.12195122 150485.87804878,
220276.582586992 150487.828697901,220274.949363179 150489.701464356,220273.226214362 150491.491836488,
220271.417291757 150493.195501133,220269.526953216 150494.808354014,220267.559752731 150496.326509628,
220265.520429459 150497.746310603,220263.41389631 150499.064336517,220261.245228106 150500.277412127,
220259.019649359 150501.38261503,220256.742521683 150502.377282695,220254.419330878 150503.259018879,
220252.055673714 150504.025699404,220249.657244448 150504.675477269,220247.229821107 150505.206787101,
220244.779251566 150505.61834893,220242.311439461 150505.909171266,220239.832329968 150506.078553494,
220237.347895479 150506.126087555,220234.864121215 150506.051658938,220232.386990804 150505.855446946,
220229.922471872 150505.537924272,220227.47650166 150505.099855856,220225.054972724 150504.542297043,
220222.663718741 150503.86659104,220220.308500449 150503.074365683,
220217.994991777 150502.167529512,220215.72876617 150501.148267175,
220213.515283163 150500.019034164,220211.35987523 150498.7825509,
220209.267734939 150497.441796181,220207.243902439 150496,
220205.293253319 150494.460635772,220203.420486864 150492.82741196,220201.630114732 150491.104263143,
220199.926450087 150489.295340538,220198.313597205 150487.405001997,220196.795441592 150485.437801511,
220195.375640616 150483.39847824,220194.057614703 150481.291945091,220192.844539092 150479.123276887,220191.739336189 150476.89769814,
220190.744668525 150474.620570464,220189.86293234 150472.297379659,220189.096251815 150469.933722495,
220188.446473951 150467.535293229,220187.915164118 150465.107869888,220187.50360229 150462.657300346,
220187.212779953 150460.189488241,220187.043397726 150457.710378749,220186.995863664 150455.22594426,
220187.070292282 150452.742169995,220187.266504273 150450.265039585,220187.584026947 150447.800520653,
220188.022095363 150445.35455044,220188.579654177 150442.933021505,220189.25536018 150440.541767521,
```

```

220190.047585536 150438.18654923,220190.954421707 150435.873040558,220191.973684044 ↵
    150433.60681495,
220193.102917055 150431.393331943,220194.339400319 150429.237924011,220195.680155039 ↵
    150427.14578372,220197.12195122 150425.12195122,
220198.661315447 150423.171302099,220200.29453926 150421.298535644,220202.017688077 ↵
    150419.508163512,220203.826610682 150417.804498867,
220205.716949223 150416.191645986,220207.684149708 150414.673490372,220209.72347298 ↵
    150413.253689397,220211.830006129 150411.935663483,
220213.998674333 150410.722587873,220216.22425308 150409.61738497,220218.501380756 ↵
    150408.622717305,220220.824571561 150407.740981121,
220223.188228725 150406.974300596,220225.586657991 150406.324522731,220227 150406)

--3d example
SELECT ST_AsEWKT(ST_CurveToLine(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 ↵
    150505 2,220227 150406 3)')));
Output
-----
LINESTRING(220268 150415 1,220269.95064912 150416.539364228 1.0181172856673,
220271.823415575 150418.17258804 1.03623457133459,220273.613787707 150419.895736857 ↵
    1.05435185700189,...AD INFINITUM ....
    220225.586657991 150406.324522731 1.32611114201132,220227 150406 3)

--use only 2 segments to approximate quarter circle
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 ↵
    150505,220227 150406)'),2));
st_astext
-----
LINESTRING(220268 150415,220287.740300149 150448.342699654,220278.12195122 ↵
    150485.87804878,
220244.779251566 150505.61834893,220207.243902439 150496,220187.50360229 150462.657300346,
220197.12195122 150425.12195122,220227 150406)

-- Ensure approximated line is no further than 20 units away from
-- original curve, and make the result direction-neutral
SELECT ST_AsText(ST_CurveToLine(
    'CIRCULARSTRING(0 0,100 -100,200 0)::geometry,
    20, -- Tolerance
    1, -- Above is max distance between curve and line
    1 -- Symmetric flag
));
st_astext
-----
LINESTRING(0 0,50 -86.6025403784438,150 -86.6025403784439,200 -1.1331077795296e-13,200 0)

```

☒☒

ST_LineToCurve

7.5.5 ST_Scroll

ST_Scroll — Change start point of a closed LineString.

Synopsis

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

国国

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.

国国

Make e closed line start at its 3rd vertex

```
SELECT ST_AseWKT(ST_Scroll('SRID=4326;LINESTRING(0 0 0 1, 10 0 2 0, 5 5 4 2,0 0 0 1)', ' ' ←
    POINT(5 5 4 2)'));
```

```
st_asewkt
```

```
-----
```

```
SRID=4326;LINESTRING(5 5 4 2,0 0 0 1,10 0 2 0,5 5 4 2)
```

国国

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);

国国

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 国国国国国国国国国国国国.



This method supports Circular Strings and Curves.



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



This function supports M coordinates.



This function supports Polyhedral surfaces.



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

例

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

例

ST_SwapOrdinates

7.5.7 ST_Force2D

ST_Force2D — 将“2 维几何”强制为 2D。

Synopsis

geometry **ST_Force2D**(geometry geomA);

例

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

2.0.0 强制 (polyhedral surface) 强制。

2.1.0 强制, 2.0.x 强制 ST_Force_2D 强制。



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



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

例

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

SELECT ST_AsEWKT(ST_Force2D('POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))'));
 st_asewkt
-----
POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))
```

函数

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);

函数

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 引入 (polyhedral surface) 支持。

版本: 2.1.0 引入, 在 2.0.x 版本中 ST_Force_3D 支持。

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

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

函数

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
          st_asewkt
-----
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)

SELECT ST_AsEWKT(ST_Force3D('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));
          st_asewkt
-----
POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

函数

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);

返回

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 添加 (polyhedral surface) 支持。

版本: 2.1.0 添加, 在 2.0.x 版本中 `ST_Force_3DZ` 未实现。

更改: 3.1.0. 添加支持供应非零 Z 值。

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

示例

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3DZ(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5
6 2)')));

                                st_asewkt
-----
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)

SELECT  ST_AsEWKT(ST_Force3DZ('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));

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

参见

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

7.5.10 ST_Force3DM

`ST_Force3DM` — 将 XYM 几何体强制转换为 3D。

Synopsis

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

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

函数: 2.1.0 添加, 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.

函数

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3DM(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5
6 2)')));
          st_asewkt
-----
CIRCULARSTRINGM(1 1 0,2 3 0,4 5 0,6 7 0,5 6 0)

SELECT ST_AsEWKT(ST_Force3DM('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1))
'));
          st_asewkt
-----
POLYGONM((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

函数

`ST_AsEWKT`, `ST_Force2D`, `ST_Force3DM`, `ST_Force3D`, `ST_GeomFromEWKT`

7.5.11 ST_Force4D

`ST_Force4D` — 函数 XYZM 函数.

Synopsis

geometry **ST_Force4D**(geometry geomA, float Zvalue = 0.0, float Mvalue = 0.0);

函数

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

函数: 2.1.0 添加, 2.0.x 函数 `ST_Force_4D` 函数.

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.

例

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force4D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));

                                st_asewkt
-----
CIRCULARSTRING(1 1 2 0,2 3 2 0,4 5 2 0,6 7 2 0,5 6 2 0)

SELECT ST_AsEWKT(ST_Force4D('MULTILINESTRINGM((0 0 1,0 5 2,5 0 3,0 0 4),(1 1 1,3 1 1,1 3 1,1 1 1))'));

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

例

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

7.5.12 ST_ForceCollection

ST_ForceCollection — 强制几何体为集合。

Synopsis

geometry **ST_ForceCollection**(geometry geomA);

例

强制几何体为集合。 返回 WKB 几何体。

2.0.0 版本 (polyhedral surface) 支持。

1.2.2 版本 (curve) 支持。 1.3.4 版本 (curve) 支持。

2.1.0 版本, 2.0.x 版本 **ST_Force_Collection** 支持。



This function supports Polyhedral surfaces.



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



This method supports Circular Strings and Curves.

例

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

国国

```
SELECT ST_AsText(
  ST_ForceCurve(
    'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))'::geometry
  )
);
```

	st_astext

	CURVEPOLYGON Z ((0 0 2,5 0 2,0 5 2,0 0 2),(1 1 2,1 3 2,3 1 2,1 1 2))
(1 row)	

国国

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);

国国

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 国国国国国国国国国国国国.

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

国国

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 新增。

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

参见

[ST_ForcePolygonCCW](#) , [ST_IsPolygonCCW](#) , [ST_IsPolygonCW](#)

7.5.16 ST_ForceSFS

ST_ForceSFS — 强制 SFS 1.1 几何体。

Synopsis

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

返回

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

7.5.17 ST_ForceRHR

ST_ForceRHR — 强制 (orientation) 几何体 (Right-Hand Rule) 右手定则。

Synopsis

geometry **ST_ForceRHR**(geometry g);

☒☒

Forces the orientation of the vertices in a polygon to follow a Right-Hand-Rule, in which the area that is bounded by the polygon is to the right of the boundary. In particular, the exterior ring is orientated in a clockwise direction and the interior rings in a counter-clockwise direction. This function is a synonym for [ST_ForcePolygonCW](#)



Note

The above definition of the Right-Hand-Rule conflicts with definitions used in other contexts. To avoid confusion, it is recommended to use [ST_ForcePolygonCW](#).

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



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



This function supports Polyhedral surfaces.

☒☒

```
SELECT ST_AsEWKT(
  ST_ForceRHR(
    'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))'
  )
);
```

	st_asewkt
POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))	
(1 row)	

☒☒

[ST_ForcePolygonCCW](#) , [ST_ForcePolygonCW](#) , [ST_IsPolygonCCW](#) , [ST_IsPolygonCW](#) , [ST_BuildArea](#), [ST_Polygonize](#), [ST_Reverse](#)

7.5.18 ST_LineExtend

ST_LineExtend — Returns a line extended forwards and backwards by specified distances.

Synopsis

geometry **ST_LineExtend**(geometry line, float distance_forward, float distance_backward=0.0);

☒☒

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

Availability: 3.4.0

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

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

[ST_LineSubstring](#), [ST_LocateAlong](#), [ST_Project](#)

7.5.19 ST_LineToCurve

ST_LineToCurve — Converts a linear geometry to a curved geometry.

Synopsis

geometry **ST_LineToCurve**(geometry geomANoncircular);

Converts plain LINESTRING/POLYGON to CIRCULAR STRINGS and Curved Polygons. Note much fewer points are needed to describe the curved equivalent.



Note If the input LINESTRING/POLYGON is not curved enough to clearly represent a curve, the function will return the same input geometry.

Availability: 1.3.0

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

```
-- 2D Example
SELECT ST_AsText(ST_LineToCurve(foo.geom)) As curvedastext,ST_AsText(foo.geom) As
non_curvedastext
FROM (SELECT ST_Buffer('POINT(1 3)::geometry, 3) As geom) As foo;

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

```

| 1.58527096604839 ↵
| 0.0576441587903094,1 ↵
| 0,
| 0.414729033951621 ↵
| 0.0576441587903077,-0.1480502 ↵
| 0.228361402466137,
| -0.666710699058802 ↵
| 0.505591163092361,-1.1213203 ↵
| 0.878679656440353,
| -1.49440883690763 ↵
| 1.33328930094119,-1.77163859 ↵
| 1.85194970290472
| --ETC-- ↵
| ,3.94235584120969 ↵
| 3.58527096604839,4 ↵
| 3))

--3D example
SELECT ST_AsText(ST_LineToCurve(geom)) As curved, ST_AsText(geom) AS not_curved
FROM (SELECT ST_Translate(ST_Force3D(ST_Boundary(ST_Buffer(ST_Point(1,3), 2,2))),0,0,3) AS
      geom) AS foo;

      curved                               | not_curved
-----+-----
CIRCULARSTRING Z (3 3 3,-1 2.999999999999999 3,3 3 3) | LINESTRING Z (3 3 3,2.4142135623731 ↵
1.58578643762691 3,1 1 3,                               | -0.414213562373092 1.5857864376269 ↵
3,-1 2.999999999999999 3,                               | -0.414213562373101 4.41421356237309 ↵
3,                                                       | 0.9999999999999991 5 ↵
3,2.41421356237309 4.4142135623731 ↵
3,3 3 3)

(1 row)
```

ST_CurveToLine

7.5.20 ST_Multi

ST_Multi —

Synopsis

geometry **ST_Multi**(geometry geom);

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

例

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

例

ST_AsText

7.5.21 ST_Normalize

ST_Normalize — 规范化几何对象。

Synopsis

geometry **ST_Normalize**(geometry geom);

例

规范化几何对象，使其符合 SQL/MM 规范。规范化后，几何对象将符合 SQL/MM 规范，并且其几何类型将保持不变。规范化后，几何对象将符合 SQL/MM 规范，并且其几何类型将保持不变。规范化后，几何对象将符合 SQL/MM 规范，并且其几何类型将保持不变。

2.3.0 版本引入。

例

```
SELECT ST_AsText(ST_Normalize(ST_GeomFromText(
  'GEOMETRYCOLLECTION(
    POINT(2 3),
    MULTILINESTRING((0 0, 1 1),(2 2, 3 3)),
    POLYGON(
      (0 10,0 0,10 0,10 10,0 10),
      (4 2,2 2,2 4,4 4,4 2),
      (6 8,8 8,8 6,6 6,6 8)
    )
  )'
)));
          st_astext
-----
GEOMETRYCOLLECTION(POLYGON((0 0,0 10,10 10,10 0,0 0),(6 6,8 6,8 8,6 8,6 6),(2 2,4 2,4 4,2 4,2 2)),MULTILINESTRING((2 2,3 3),(0 0,1 1)),POINT(2 3))
(1 row)
```

例

ST_Equals,

7.5.22 ST_Project

ST_Project — Returns a point projected from a start point by a distance and bearing (azimuth).

Synopsis

```
geometry ST_Project(geometry g1, float distance, float azimuth);
geometry ST_Project(geometry g1, geometry g2, float distance);
geography ST_Project(geography g1, float distance, float azimuth);
geography ST_Project(geography g1, geography g2, float distance);
```

描述

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 添加对负距离的支持。

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)
```

参见

[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);

国国

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.

国国

```
SELECT ST_AsText(ST_QuantizeCoordinates('POINT (100.123456 0)::geometry, 4));
st_astext
-----
POINT(100.123455047607 0)
```

```
WITH test AS (SELECT 'POINT (123.456789123456 123.456789123456)::geometry AS geom)
SELECT
  digits,
  encode(ST_QuantizeCoordinates(geom, digits), 'hex'),
  ST_AsText(ST_QuantizeCoordinates(geom, digits))
FROM test, generate_series(15, -15, -1) AS digits;
```

digits	encode	st_astext
15	010100000005f9a72083cdd5e405f9a72083cdd5e40	POINT(123.456789123456 123.456789123456)
14	010100000005f9a72083cdd5e405f9a72083cdd5e40	POINT(123.456789123456 123.456789123456)
13	010100000005f9a72083cdd5e405f9a72083cdd5e40	POINT(123.456789123456 123.456789123456)
12	010100000005c9a72083cdd5e405c9a72083cdd5e40	POINT(123.456789123456 123.456789123456)
11	01010000000409a72083cdd5e40409a72083cdd5e40	POINT(123.456789123456 123.456789123456)
10	0101000000009a72083cdd5e40009a72083cdd5e40	POINT(123.456789123455 123.456789123455)
9	0101000000009072083cdd5e40009072083cdd5e40	POINT(123.456789123418 123.456789123418)
8	0101000000008072083cdd5e40008072083cdd5e40	POINT(123.45678912336 123.45678912336)
7	0101000000000070083cdd5e40000070083cdd5e40	POINT(123.456789121032 123.456789121032)
6	0101000000000040083cdd5e40000040083cdd5e40	POINT(123.456789076328 123.456789076328)
5	0101000000000000083cdd5e400000000083cdd5e40	POINT(123.456789016724 123.456789016724)
4	0101000000000000003cdd5e400000000003cdd5e40	POINT(123.456787109375 123.456787109375)
3	010100000000000000003cdd5e400000000003cdd5e40	POINT(123.456787109375 123.456787109375)
2	0101000000000000000038dd5e4000000000038dd5e40	POINT(123.45654296875 123.45654296875)
1	01010000000000000000dd5e40000000000dd5e40	POINT(123.453125 123.453125)
0	01010000000000000000dc5e40000000000dc5e40	POINT(123.4375 123.4375)
-1	01010000000000000000c05e400000000000c05e40	POINT(123 123)
-2	0101000000000000000005e400000000000005e40	POINT(120 120)
-3	01010000000000000000058400000000000005840	POINT(96 96)
-4	01010000000000000000058400000000000005840	POINT(96 96)
-5	01010000000000000000058400000000000005840	POINT(96 96)
-6	01010000000000000000058400000000000005840	POINT(96 96)
-7	01010000000000000000058400000000000005840	POINT(96 96)
-8	01010000000000000000058400000000000005840	POINT(96 96)
-9	01010000000000000000058400000000000005840	POINT(96 96)
-10	01010000000000000000058400000000000005840	POINT(96 96)
-11	01010000000000000000058400000000000005840	POINT(96 96)
-12	01010000000000000000058400000000000005840	POINT(96 96)
-13	01010000000000000000058400000000000005840	POINT(96 96)
-14	01010000000000000000058400000000000005840	POINT(96 96)
-15	01010000000000000000058400000000000005840	POINT(96 96)



ST_SnapToGrid

7.5.24 ST_RemovePoint

ST_RemovePoint — Remove a point from a linestring.

Synopsis

geometry **ST_RemovePoint**(geometry linestring, integer offset);

⚠

Removes a point from a LineString, given its index (0-based). Useful for turning a closed line (ring) into an open linestring.

Enhanced: 3.2.0

1.1.0 函数参考.



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

⚠

Guarantees no lines are closed by removing the end point of closed lines (rings). Assumes geom is of type LINESTRING

```
UPDATE sometable
  SET geom = ST_RemovePoint(geom, ST_NPoints(geom) - 1)
FROM sometable
WHERE ST_IsClosed(geom);
```

⚠

[ST_AddPoint](#), [ST_NPoints](#), [ST_NumPoints](#)

7.5.25 ST_RemoveRepeatedPoints

ST_RemoveRepeatedPoints — Returns a version of a geometry with duplicate points removed.

Synopsis

geometry **ST_RemoveRepeatedPoints**(geometry geom, float8 tolerance = 0.0);

⚠

Returns a version of the given geometry with duplicate consecutive points removed. The function processes only (Multi)LineStrings, (Multi)Polygons and MultiPoints but it can be called with any kind of geometry. Elements of GeometryCollections are processed individually. The endpoints of LineStrings are preserved.

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

Enhanced: 3.2.0

2.2.0 函数参数.

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

SQL

```
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)
```

SQL

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);

SQL

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

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

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

In contrast to `ST_ClipByBox2D()` this function

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

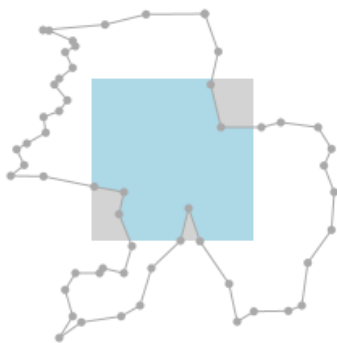
If `cartesian_hint` is set to `true`, the algorithm applies additional optimizations involving cartesian math to further reduce the resulting point number. Please note that using this option might introduce rendering artifacts if the resulting coordinates are projected into another (non-cartesian) coordinate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



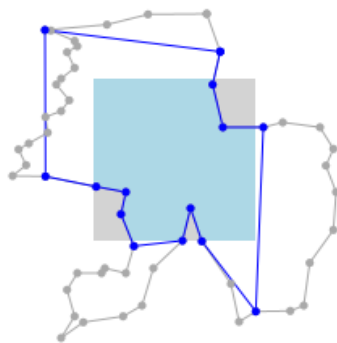
Warning

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

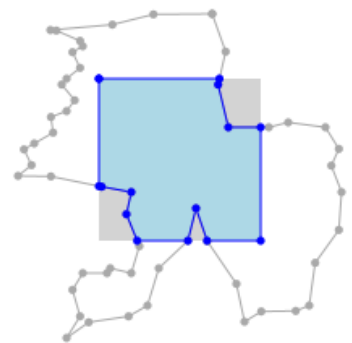
original
55 points



`ST_RemoveIrrelevantPointsForView(geom, bbox)`
15 points

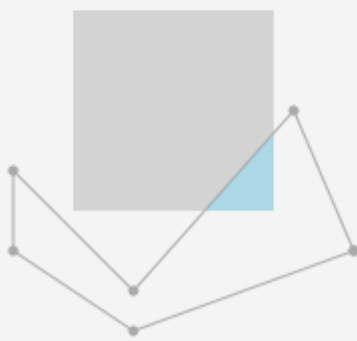


`ST_ClipByBox2D(geom, bbox)`
15 points

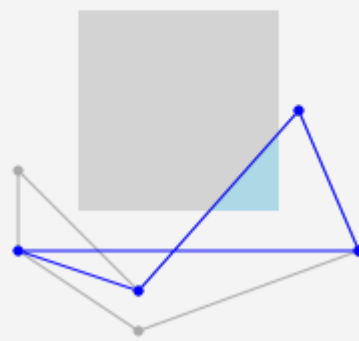


Example: `ST_RemoveIrrelevantPointsForView()` applied to a polygon. Blue points remain, the rendering result (light-blue area) within the grey view box remains as well.

original
7 points



`ST_RemoveIrrelevantPointsForView(geom, bbox)`
5 points



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

Availability: 3.5.0

实验

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

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

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

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

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

st_astext
-----
LINESTRING EMPTY
```

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

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

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

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

实验

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);

国国

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

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

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

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

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

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

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

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

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

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

original
20 points, 4 parts



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



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



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

Availability: 3.5.0

例

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

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

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

st_astext
-----
LINESTRING EMPTY
```

7.5.28 ST_Reverse

ST_Reverse — 返回几何对象的反向几何。

Synopsis

geometry **ST_Reverse**(geometry g1);

新特性

ST_MakeLine, ST_MakeLineFromPoints.

Enhanced: 2.4.0 support for curves was introduced.



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



This function supports Polyhedral surfaces.

新特性

```
SELECT ST_AsText(geom) as line, ST_AsText(ST_Reverse(geom)) As reverseline
FROM
(SELECT ST_MakeLine(ST_Point(1,2),
                    ST_Point(1,10)) As geom) as foo;
--result
      line      |      reverseline
-----+-----
LINESTRING(1 2,1 10) | LINESTRING(1 10,1 2)
```

7.5.29 ST_Segmentize

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

Synopsis

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

新特性

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 函数 `ST_Segmentize`

Enhanced: 3.0.0 Segmentize geometry now produces equal-length subsegments

Enhanced: 2.3.0 Segmentize geography now produces equal-length subsegments

新增: 2.1.0 函数 `ST_Segmentize` 支持地理类型。

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)`)

描述

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

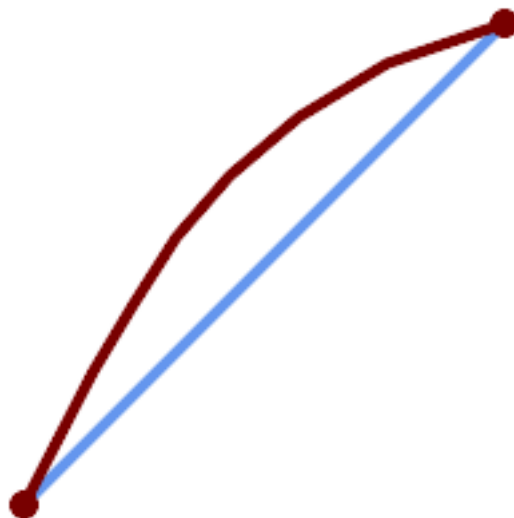
```
SELECT ST_AsText(ST_Segmentize(
  'MULTILINESTRING((0 0, 0 1, 0 9),(1 10, 1 18))'::geometry,
    5 ) );
-----
MULTILINESTRING((0 0,0 1,0 5,0 9),(1 10,1 14,1 18))
```

Segmentizing a polygon:

```
SELECT ST_AsText(
  ST_Segmentize(('POLYGON((0 0, 0 8, 30 0, 0 0))'::geometry), 10));
-----
POLYGON((0 0,0 8,7.5 6,15 4,22.5 2,30 0,20 0,10 0,0 0))
```

Segmentizing a geographic line, using a maximum segment length of 2000 kilometers. Vertices are added along the great-circle arc connecting the endpoints.

```
SELECT ST_AsText(
  ST_Segmentize(('LINESTRING (0 0, 60 60)'::geography), 2000000));
-----
LINESTRING(0 0,4.252632294621186 8.43596525986862,8.69579947419404 ↵
  16.824093489701564,13.550465473227048 25.107950473646188,19.1066053508691 ↵
  33.21091076089908,25.779290201459894 41.01711439406505,34.188839517966954 ↵
  48.337222885886,45.238153936612264 54.84733442373889,60 60)
```



A geographic line segmentized along a great circle arc

☐☐

[ST_LineSubstring](#)

7.5.30 ST_SetPoint

ST_SetPoint — 将点插入到线串中。

Synopsis

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

☐☐

将点插入到线串中。N 是插入点的位置。0 表示第一个点。-1 表示最后一个点。如果 N 是负数，则从最后一个点开始计数。如果 N 是正数，则从第一个点开始计数。如果 N 是 0，则插入到第一个点之前。如果 N 是 -1，则插入到最后一个点之后。

1.1.0 版本引入。

更新到: 2.3.0 版本。



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

☐☐

```
--Change first point in line string from -1 3 to -1 1
SELECT ST_AsText(ST_SetPoint('LINESTRING(-1 2,-1 3)', 0, 'POINT(-1 1)'));
          st_astext
-----
LINESTRING(-1 1,-1 3)

---Change last point in a line string (lets play with 3d linestring this time)
SELECT ST_AsEWKT(ST_SetPoint(foo.geom, ST_NumPoints(foo.geom) - 1, ST_GeomFromEWKT('POINT (-1 1 3)'))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(-1 2 3,-1 3 4, 5 6 7)') As geom) As foo;
          st_asewkt
-----
LINESTRING(-1 2 3,-1 3 4,-1 1 3)

SELECT ST_AsText(ST_SetPoint(g, -3, p))
FROM ST_GeomFromText('LINESTRING(0 0, 1 1, 2 2, 3 3, 4 4)') AS g
      , ST_PointN(g,1) as p;
          st_astext
-----
LINESTRING(0 0,1 1,0 0,3 3,4 4)
```

☐☐

[ST_AddPoint](#), [ST_NPoints](#), [ST_NumPoints](#), [ST_PointN](#), [ST_RemovePoint](#)

7.5.31 ST_ShiftLongitude

ST_ShiftLongitude — Shifts the longitude coordinates of a geometry between -180..180 and 0..360.

Synopsis

geometry **ST_ShiftLongitude**(geometry geom);

返回

Reads every point/vertex in a geometry, and shifts its longitude coordinate from -180..0 to 180..360 and vice versa if between these ranges. This function is symmetrical so the result is a 0..360 representation of a -180..180 data and a -180..180 representation of a 0..360 data.



Note

This is only useful for data with coordinates in longitude/latitude; e.g. SRID 4326 (WGS 84 geographic)



Warning

1.3.4 版本中，该函数仅支持 2D 几何。1.3.4 版本中，该函数仅支持 2D 几何。



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

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

版本: 2.2.0 版本, 使用 "ST_Shift_Longitude" 函数。



This function supports Polyhedral surfaces.



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

返回

```
--single point forward transformation
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(270 0)::geometry'))
```

```
st_astext
-----
POINT(-90 0)
```

```
--single point reverse transformation
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(-90 0)::geometry'))
```

```
st_astext
-----
POINT(270 0)
```

```
--for linestrings the functions affects only to the sufficient coordinates
```

```
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;LINESTRING(174 12, 182 13)::geometry'))
st_astext
-----
LINESTRING(174 12,-178 13)
```

☐☐

ST_WrapX

7.5.32 ST_WrapX

ST_WrapX — X 坐标的地理坐标系统。

Synopsis

geometry **ST_WrapX**(geometry geom, float8 wrap, float8 move);

☐☐

This function splits the input geometries and then moves every resulting component falling on the right (for negative 'move') or on the left (for positive 'move') of given 'wrap' line in the direction specified by the 'move' parameter, finally re-unioning the pieces together.



Note

ST_WrapX 函数在 GEOS 3.10.0 之前是不可用的。

Availability: 2.3.0 requires GEOS



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

☐☐

```
-- Move all components of the given geometries whose bounding box
-- falls completely on the left of x=0 to +360
select ST_WrapX(geom, 0, 360);

-- Move all components of the given geometries whose bounding box
-- falls completely on the left of x=-30 to +360
select ST_WrapX(geom, -30, 360);
```

☐☐

ST_ShiftLongitude

7.5.33 ST_SnapToGrid

ST_SnapToGrid — 将几何体 (geom) 对齐到指定的精度网格 (snap)。

Synopsis

```
geometry ST_SnapToGrid(geometry geomA, float originX, float originY, float sizeX, float sizeY);
geometry ST_SnapToGrid(geometry geomA, float sizeX, float sizeY);
geometry ST_SnapToGrid(geometry geomA, float size);
geometry ST_SnapToGrid(geometry geomA, geometry pointOrigin, float sizeX, float sizeY, float sizeZ,
float sizeM);
```

注意

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

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



Note

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



Note

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

1.0.0RC1 指定的精度网格指定的精度网格。

1.1.0 指定的精度网格 Z 指定的精度网格 M 指定的精度网格。

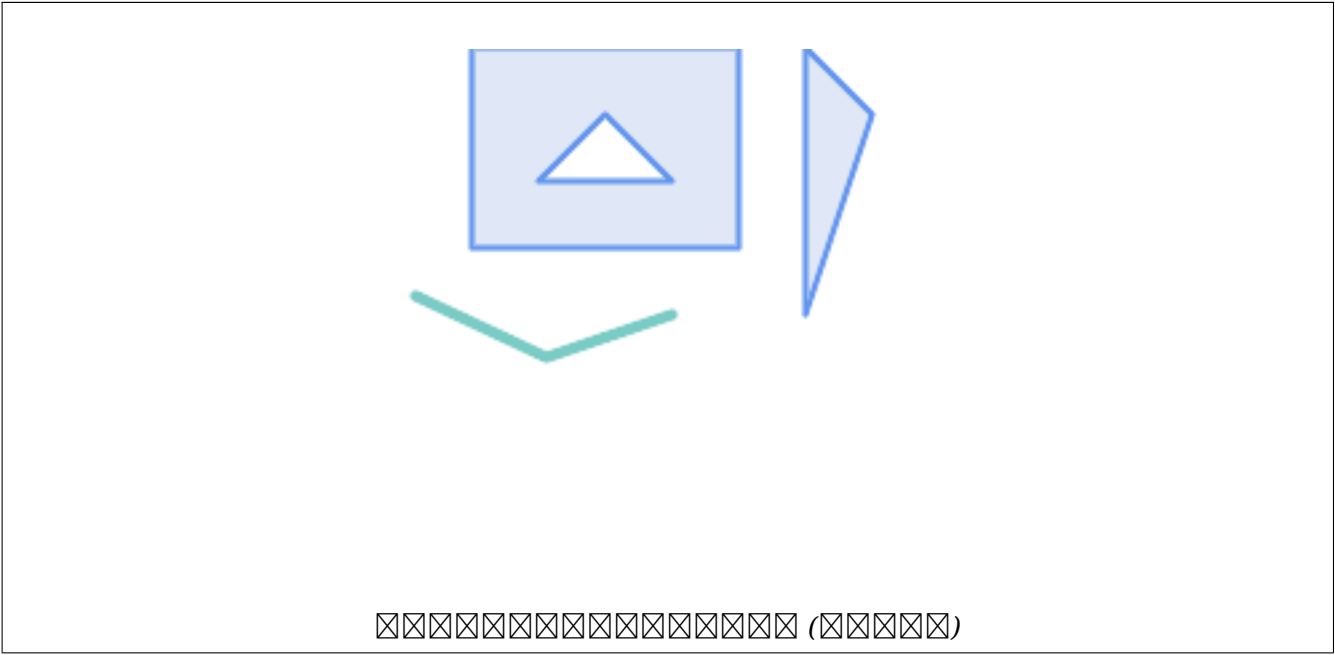


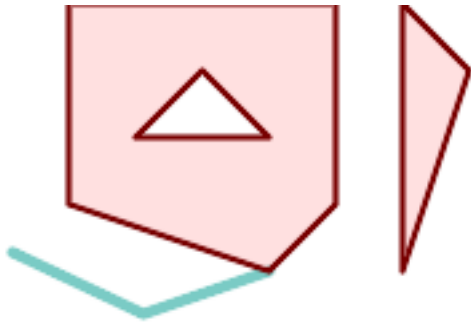
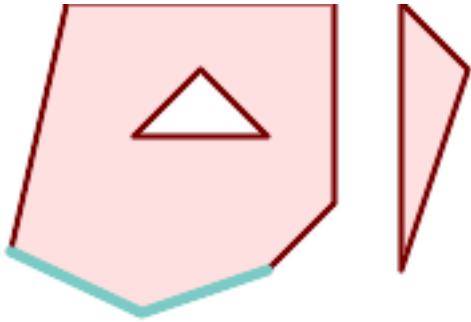
This function supports 3d and will not drop the z-index.

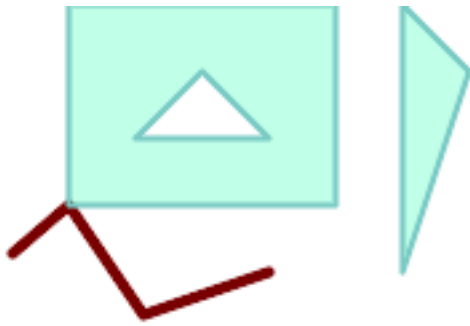
注意

```
--Snap your geometries to a precision grid of 10^-3
UPDATE mytable
  SET geom = ST_SnapToGrid(geom, 0.001);

SELECT ST_AsText(ST_SnapToGrid(
    ST_GeomFromText('LINESTRING(1.1115678 2.123, 4.111111 3.2374897, ↵
    4.11112 3.23748667)'),
    0.001)
    );
           st_astext
-----
LINESTRING(1.112 2.123,4.111 3.237)
--Snap a 4d geometry
SELECT ST_AsEWKT(ST_SnapToGrid(
```

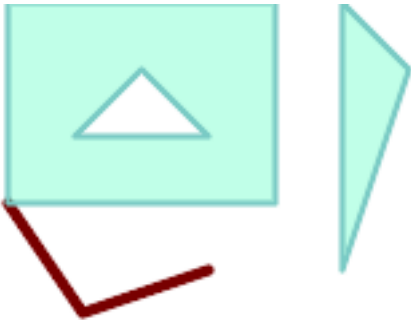
	
距离 1.01 将线段 snapping 到多边形边界上。 距离 1.25 将线段 snapping 到多边形边界上。 <pre>SELECT ST_AsText(ST_Snap(poly,line, ST_Distance(poly,line)*1.01)) AS polysnapped FROM (SELECT ST_GeomFromText('MULTIPOLYGON(((26 125, 26 200, 126 200, 126 125, 26 125), (51 150, 101 150, 76 175, 51 150)), ((151 100, 151 200, 176 175, 151 100)))') As poly, ST_GeomFromText('LINESTRING (5 107, 54 84, 101 100)') As line) As foo;</pre> polysnapped	<pre>SELECT ST_AsText(ST_Snap(poly,line, ST_Distance(poly, line)*1.25)) AS polysnapped FROM (SELECT ST_GeomFromText('MULTIPOLYGON(((26 125, 26 200, 126 200, 126 125, 26 125), (51 150, 101 150, 76 175, 51 150)), ((151 100, 151 200, 176 175, 151 100)))') As poly, ST_GeomFromText('LINESTRING (5 107, 54 84, 101 100)') As line) As foo;</pre> polysnapped
<pre>MULTIPOLYGON(((26 125,26 200,126 200,126 125,101 100,26 125), (51 150,101 150,76 175,51 150)),((151 100,151 200,176 175,151 100)))</pre>	<pre>MULTIPOLYGON(((5 107,26 200,126 200,126 125,101 100,54 84,5 107), (51 150,101 150,76 175,51 150)),((151 100,151 200,176 175,151 100)))</pre>



距离 1.01 将红色线段的端点移动到最近的青色多边形边界上。图中展示了原始位置（左侧）和 snapped 后的位置（右侧）。

```
SELECT ST_AsText(
  ST_Snap(line, poly, ST_Distance(poly, line)*1.01)
) AS linesnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
    ((26 125, 26 200, 126 200, 126 125,
    26 125),
    (51 150, 101 150, 76 175, 51 150 ))
  ',
    ((151 100, 151 200, 176 175, 151
    100)))') As poly,
    ST_GeomFromText('LINESTRING (5
    107, 54 84, 101 100)') As line
  ) As foo;

          linesnapped
-----
LINESTRING(5 107,26 125,54 84,101 100)
```



距离 1.25 将红色线段的端点移动到最近的青色多边形边界上。图中展示了原始位置（左侧）和 snapped 后的位置（右侧）。

```
SELECT ST_AsText(
  ST_Snap(line, poly, ST_Distance(poly, line)*1.25)
) AS linesnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
    ( 26 125, 26 200, 126 200, 126 125,
    26 125 ),
    (51 150, 101 150, 76 175, 51 150 ))
  ',
    ((151 100, 151 200, 176 175, 151
    100 )))') As poly,
    ST_GeomFromText('LINESTRING (5
    107, 54 84, 101 100)') As line
  ) As foo;

          linesnapped
-----
LINESTRING(26 125,54 84,101 100)
```

☒

ST_SnapToGrid

7.5.35 ST_SwapOrdinates

ST_SwapOrdinates — 交换几何体中的两个有序坐标值。

Synopsis

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

函数

函数返回一个文本字符串。

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

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

函数

```
-- Scale M value by 2
SELECT ST_AsText(
  ST_SwapOrdinates(
    ST_Scale(
      ST_SwapOrdinates(g, 'xm'),
      2, 1
    ),
    'xm'
  ) FROM ( SELECT 'POINT ZM (0 0 0 2)::geometry g ) foo;
      st_astext
-----
POINT ZM (0 0 0 4)
```

函数

ST_FlipCoordinates

7.6 Geometry Validation

7.6.1 ST_IsValid

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

Synopsis

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


☒☒

Tests if an `ST_Geometry` value is well-formed and valid in 2D according to the OGC rules. For geometries with 3 and 4 dimensions, the validity is still only tested in 2 dimensions. For geometries that are invalid, a PostgreSQL NOTICE is emitted providing details of why it is not valid.

For the version with the `flags` parameter, supported values are documented in [ST_IsValidDetail](#). This version does not print a NOTICE explaining invalidity.

For more information on the definition of geometry validity, refer to [Section 4.4](#).



Note

SQL-MM defines the result of `ST_IsValid(NULL)` to be 0, while PostGIS returns NULL.

GEOS ☒☒☒☒☒

The version accepting flags is available starting with 2.0.0.



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



This method implements the SQL/MM specification. SQL-MM 3: 5.1.9



Note

Neither OGC-SFS nor SQL-MM specifications include a flag argument for `ST_IsValid`. The flag is a PostGIS extension.

☒☒

```
SELECT ST_IsValid(ST_GeomFromText('LINESTRING(0 0, 1 1)')) As good_line,
       ST_IsValid(ST_GeomFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) As bad_poly
--results
NOTICE: Self-intersection at or near point 0 0
good_line | bad_poly
-----+-----
t         | f
```

☒☒

[ST_IsSimple](#), [ST_IsValidReason](#), [ST_IsValidDetail](#),

7.6.2 ST_IsValidDetail

`ST_IsValidDetail` — Returns a `valid_detail` row stating if a geometry is valid or if not a reason and a location.

Synopsis

`valid_detail` **`ST_IsValidDetail`**(geometry geom, integer flags);

国国

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 国国国国国

2.0.0 国国国国国国国国国国国国.

国国

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

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

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

valid	reason	location
t		

☐☐

[ST_IsValid](#), [ST_IsValidReason](#)

7.6.3 ST_IsValidReason

[ST_IsValidReason](#) — Returns text stating if a geometry is valid, or a reason for invalidity.

Synopsis

```
text ST_IsValidReason(geometry geomA);
text ST_IsValidReason(geometry geomA, integer flags);
```

☐☐

Returns text stating if a geometry is valid, or if invalid a reason why.

Useful in combination with [ST_IsValid](#) to generate a detailed report of invalid geometries and reasons.

Allowed flags are documented in [ST_IsValidDetail](#).

GEOS ☐☐☐☐☐

Availability: 1.4

Availability: 2.0 version taking flags.

☐☐

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

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

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

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

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

st_isvalidreason
Valid Geometry

☒☒

ST_IsValid, ST_Summary

7.6.4 ST_MakeValid

ST_MakeValid — Attempts to make an invalid geometry valid without losing vertices.

Synopsis

```
geometry ST_MakeValid(geometry input);
geometry ST_MakeValid(geometry input, text params);
```

☒☒

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

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

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

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

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

The "method" key has two values.

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

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

GEOS 

2.0.0 .

Enhanced: 2.0.1, speed improvements

Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT.

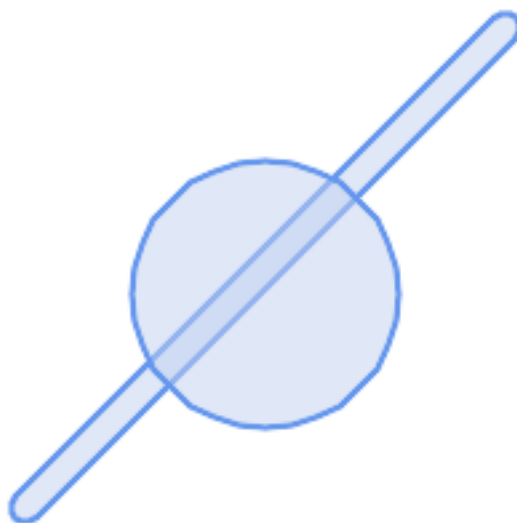
Enhanced: 3.1.0, added removal of Coordinates with NaN values.

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

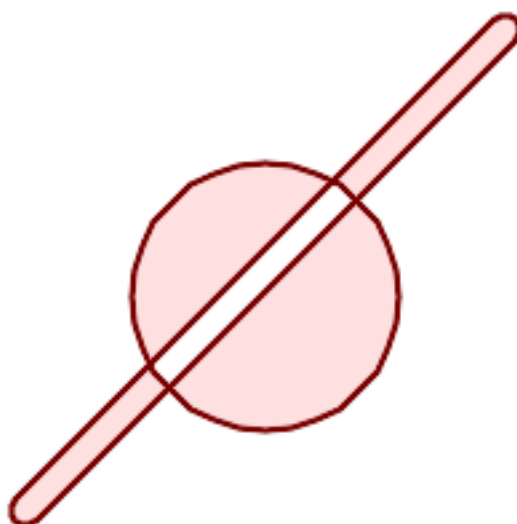


This function supports 3d and will not drop the z-index.

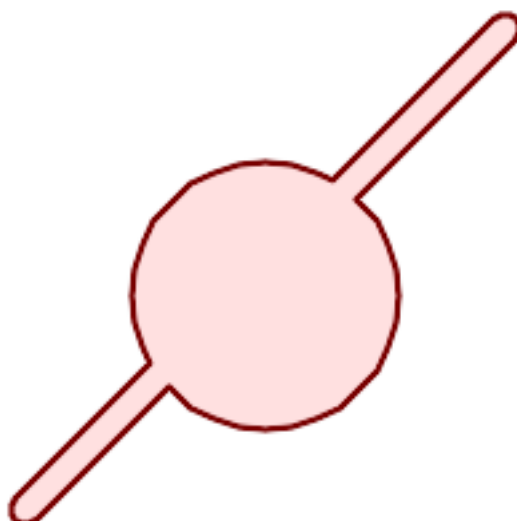




before_geom: MULTIPOLYGON of 2 overlapping polygons

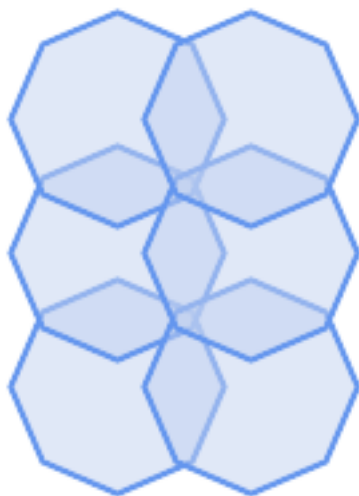


after_geom: MULTIPOLYGON of 4 non-overlapping polygons

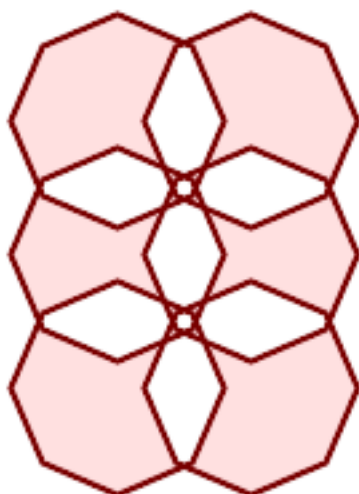


after_geom_structure: MULTIPOLYGON of 1 non-overlapping polygon

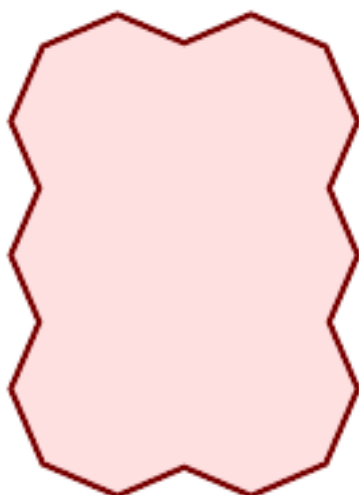
```
SELECT f.geom AS before_geom, ST_MakeValid(f.geom) AS after_geom, ST_MakeValid(f.geom, ↵
    'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,
```



before_geom: MULTIPOLYGON of 6 overlapping polygons



after_geom: MULTIPOLYGON of 14 Non-overlapping polygons



after_geom_structure: MULTIPOLYGON of 1 Non-overlapping polygon

```
SELECT c.geom AS before_geom,  
       ST_MakeValid(c.geom) AS after_geom,  
       ST_MakeValid(c.geom, 'method=structure') AS after_geom_structure  
FROM (SELECT 'MULTIPOLYGON(((91 50.79 22.51 10.23 22.11 50.23 78.51 90.79 78.91
```

 例

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

st_astext
-----
POINT(0 0)

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

st_astext
-----
LINESTRING EMPTY
```

例

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

7.7 Spatial Reference System Functions

7.7.1 ST_InverseTransformPipeline

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

Synopsis

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

例

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

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

Availability: 3.4.0

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

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



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

```
-- Inverse direction
SELECT ST_AsText(ST_InverseTransformPipeline('POINT(426857.9877165967 5427937.523342293)')::geometry,
    'urn:ogc:def:coordinateOperation:EPSG::16031')) AS wgs_geom;

-----
POINT(2 48.99999999999999)
(1 row)
```

GDA2020 example.

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

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



ST_Transform, **ST_TransformPipeline**

7.7.2 ST_SetSRID

ST_SetSRID — Set the SRID on a geometry.

Synopsis

geometry **ST_SetSRID**(geometry geom, integer srid);



Sets the SRID on a geometry to a particular integer value. Useful in constructing bounding boxes for queries.



Note

This function does not transform the geometry coordinates in any way - it simply sets the meta data defining the spatial reference system the geometry is assumed to be in. Use **ST_Transform** if you want to transform the geometry into a new projection.



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



This method supports Circular Strings and Curves.

❏

-- Mark a point as WGS 84 long lat --

```
SELECT ST_SetSRID(ST_Point(-123.365556, 48.428611),4326) As wgs84long_lat;
-- the ewkt representation (wrap with ST_AsEWKT) -
SRID=4326;POINT(-123.365556 48.428611)
```

-- Mark a point as WGS 84 long lat and then transform to web mercator (Spherical Mercator) --

```
SELECT ST_Transform(ST_SetSRID(ST_Point(-123.365556, 48.428611),4326),3785) As spere_merc;
-- the ewkt representation (wrap with ST_AsEWKT) -
SRID=3785;POINT(-13732990.8753491 6178458.96425423)
```

❏

Section [4.5](#), [ST_SRID](#), [ST_Transform](#), [UpdateGeometrySRID](#)

7.7.3 ST_SRID

ST_SRID — Returns the spatial reference identifier for a geometry.

Synopsis

integer **ST_SRID**(geometry g1);

❏

Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table. Section [4.5](#)



Note

spatial_ref_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries.



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



This method implements the SQL/MM specification. SQL-MM 3: 5.1.5



This method supports Circular Strings and Curves.

❏

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
-- result
4326
```



Section [4.5](#), [ST_SetSRID](#), [ST_Transform](#), [ST_SRID](#), [ST_SRID](#)

7.7.4 ST_Transform

ST_Transform — Return a new geometry with coordinates transformed to a different spatial reference system.

Synopsis

```
geometry ST_Transform(geometry g1, integer srid);
geometry ST_Transform(geometry geom, text to_proj);
geometry ST_Transform(geometry geom, text from_proj, text to_proj);
geometry ST_Transform(geometry geom, text from_proj, integer to_srid);
```



Returns a new geometry with its coordinates transformed to a different spatial reference system. The destination spatial reference `to_srid` may be identified by a valid SRID integer parameter (i.e. it must exist in the `spatial_ref_sys` table). Alternatively, a spatial reference defined as a PROJ.4 string can be used for `to_proj` and/or `from_proj`, however these methods are not optimized. If the destination spatial reference system is expressed with a PROJ.4 string instead of an SRID, the SRID of the output geometry will be set to zero. With the exception of functions with `from_proj`, input geometries must have a defined SRID.

`ST_Transform` is often confused with [ST_SetSRID](#). `ST_Transform` actually changes the coordinates of a geometry from one spatial reference system to another, while `ST_SetSRID()` simply changes the SRID identifier of the geometry.

`ST_Transform` automatically selects a suitable conversion pipeline given the source and target spatial reference systems. To use a specific conversion method, use [ST_TransformPipeline](#).



Note

Requires PostGIS be compiled with PROJ support. Use [PostGIS_Full_Version](#) to confirm you have PROJ support compiled in.

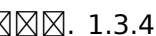


Note

If using more than one transformation, it is useful to have a functional index on the commonly used transformations to take advantage of index usage.



Note

1.3.4  (curve) . 1.3.4 .

: 2.0.0  (polyhedral surface) .

Enhanced: 2.3.0 support for direct PROJ.4 text was introduced.

- ☑ This method implements the SQL/MM specification. SQL-MM 3: 5.1.6
- ☑ This method supports Circular Strings and Curves.
- ☑ This function supports Polyhedral surfaces.

☒☒

Change Massachusetts state plane US feet geometry to WGS 84 long lat

```
SELECT ST_AsText(ST_Transform(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
743265 2967450,743265.625 2967416,743238 2967416))',2249),4326)) As wgs_geom;

wgs_geom
-----
POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.177684
8522251 42.3902896512902));
(1 row)

--3D Circular String example
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromEWKT('SRID=2249;CIRCULARSTRING(743238 2967416 ↵
1,743238 2967450 2,743265 2967450 3,743265.625 2967416 3,743238 2967416 4)'),4326));

st_asewkt
-----
SRID=4326;CIRCULARSTRING(-71.1776848522251 42.3902896512902 1,-71.1776843766326 ↵
42.3903829478009 2,
-71.1775844305465 42.3903826677917 3,
-71.1775825927231 42.3902893647987 3,-71.1776848522251 42.3902896512902 4)
```

Example of creating a partial functional index. For tables where you are not sure all the geometries will be filled in, its best to use a partial index that leaves out null geometries which will both conserve space and make your index smaller and more efficient.

```
CREATE INDEX idx_geom_26986_parcel
ON parcels
USING gist
(ST_Transform(geom, 26986))
WHERE geom IS NOT NULL;
```

Examples of using PROJ.4 text to transform with custom spatial references.

```
-- Find intersection of two polygons near the North pole, using a custom Gnomonic projection
-- See http://boundlessgeo.com/2012/02/flattening-the-peel/
WITH data AS (
  SELECT
    ST_GeomFromText('POLYGON((170 50,170 72,-130 72,-130 50,170 50))', 4326) AS p1,
    ST_GeomFromText('POLYGON((-170 68,-170 90,-141 90,-141 68,-170 68))', 4326) AS p2,
    'proj=gnom +ellps=WGS84 +lat_0=70 +lon_0=-160 +no_defs'::text AS gnom
)
SELECT ST_AsText(
  ST_Transform(
    ST_Intersection(ST_Transform(p1, gnom), ST_Transform(p2, gnom)),
    gnom, 4326))
FROM data;

st_astext
-----
```

```
POLYGON((-170 74.053793645338,-141 73.4268621378904,-141 68,-170 68,-170 74.053793645338) ↵
)
```

Configuring transformation behavior

Sometimes coordinate transformation involving a grid-shift can fail, for example if PROJ.4 has not been built with grid-shift files or the coordinate does not lie within the range for which the grid shift is defined. By default, PostGIS will throw an error if a grid shift file is not present, but this behavior can be configured on a per-SRID basis either by testing different `to_proj` values of PROJ.4 text, or altering the `proj4text` value within the `spatial_ref_sys` table.

For example, the `proj4text` parameter `+datum=NAD87` is a shorthand form for the following `+nadgrids` parameter:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat
```

The `@` prefix means no error is reported if the files are not present, but if the end of the list is reached with no file having been appropriate (ie. found and overlapping) then an error is issued.

If, conversely, you wanted to ensure that at least the standard files were present, but that if all files were scanned without a hit a null transformation is applied you could use:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat,null
```

The null grid shift file is a valid grid shift file covering the whole world and applying no shift. So for a complete example, if you wanted to alter PostGIS so that transformations to SRID 4267 that didn't lie within the correct range did not throw an ERROR, you would use the following:

```
UPDATE spatial_ref_sys SET proj4text = '+proj=longlat +ellps=clrk66 +nadgrids=@conus, ↵
@alaska,@ntv2_0.gsb,@ntv1_can.dat,null +no_defs' WHERE srid = 4267;
```



Section [4.5](#), [ST_SetSRID](#), [ST_SRID](#), [UpdateGeometrySRID](#), [ST_TransformPipeline](#)

7.7.5 ST_TransformPipeline

ST_TransformPipeline — Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.

Synopsis

```
geometry ST_TransformPipeline(geometry g1, text pipeline, integer to_srid);
```



Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.

Transformation pipelines are defined using any of the following string formats:

- `urn:ogc:def:coordinateOperation:AUTHORITY::CODE`. Note that a simple `EPSG:CODE` string does not uniquely identify a coordinate operation: the same EPSG code can be used for a CRS definition.

- A PROJ pipeline string of the form: `+proj=pipeline` ... Automatic axis normalisation will not be applied, and if necessary the caller will need to add an additional pipeline step, or remove axis swap steps.
- Concatenated operations of the form: `urn:ogc:def:coordinateOperation,coordinateOperation:EPSG:`

Availability: 3.4.0

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

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

国国

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

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

                utm_geom
-----
POINT(426857.9877165967 5427937.523342293)
(1 row)

-- Inverse direction
SELECT ST_AsText(ST_InverseTransformPipeline('POINT(426857.9877165967 5427937.523342293)::geometry,
    'urn:ogc:def:coordinateOperation:EPSG::16031')) AS wgs_geom;

                wgs_geom
-----
POINT(2 48.999999999999999)
(1 row)
```

GDA2020 example.

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

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

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

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

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

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

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

☐☐

[ST_Transform](#), [ST_InverseTransformPipeline](#)

7.7.6 postgis_srs_codes

`postgis_srs_codes` — Return the list of SRS codes associated with the given authority.

Synopsis

setof text **postgis_srs_codes**(text auth_name);

☐☐

Returns a set of all auth_srid for the given auth_name.

Availability: 3.4.0

Proj version 6+

☐☐

List the first ten codes associated with the EPSG authority.

```
SELECT * FROM postgis_srs_codes('EPSG') LIMIT 10;
```

```
postgis_srs_codes
-----
2000
20004
20005
20006
20007
20008
20009
2001
20010
20011
```

☐☐

[postgis_srs](#), [postgis_srs_all](#), [postgis_srs_search](#)

7.7.7 postgis_srs

`postgis_srs` — Return a metadata record for the requested authority and srid.

Synopsis

setof record **postgis_srs**(text auth_name, text auth_srid);

￼

Returns a metadata record for the requested `auth_srid` for the given `auth_name`. The record will have the `auth_name`, `auth_srid`, `sname`, `srttext`, `proj4text`, and the corners of the area of usage, `point_sw` and `point_ne`.

Availability: 3.4.0

Proj version 6+

￼

Get the metadata for EPSG:3005.

```
SELECT * FROM postgis_srs('EPSG', '3005');
```

```
auth_name | EPSG
auth_srid | 3005
sname     | NAD83 / BC Albers
srttext   | PROJCS["NAD83 / BC Albers", ... ]
proj4text | +proj=aea +lat_0=45 +lon_0=-126 +lat_1=50 +lat_2=58.5 +x_0=1000000 +y_0=0 +
      datum=NAD83 +units=m +no_defs +type=crs
point_sw  | 0101000020E6100000E17A14AE476161C000000000000204840
point_ne  | 0101000020E610000085EB51B81E855CC0E17A14AE47014E40
```

￼

[postgis_srs_codes](#), [postgis_srs_all](#), [postgis_srs_search](#)

7.7.8 postgis_srs_all

`postgis_srs_all` — Return metadata records for every spatial reference system in the underlying Proj database.

Synopsis

setof record **postgis_srs_all**(void);

￼

Returns a set of all metadata records in the underlying Proj database. The records will have the `auth_name`, `auth_srid`, `sname`, `srttext`, `proj4text`, and the corners of the area of usage, `point_sw` and `point_ne`.

Availability: 3.4.0

Proj version 6+

❏❏

Get the first 10 metadata records from the Proj database.

```
SELECT auth_name, auth_srid, sname FROM postgis_srs_all() LIMIT 10;
```

auth_name	auth_srid	sname
EPSG	2000	Anguilla 1957 / British West Indies Grid
EPSG	20004	Pulkovo 1995 / Gauss-Kruger zone 4
EPSG	20005	Pulkovo 1995 / Gauss-Kruger zone 5
EPSG	20006	Pulkovo 1995 / Gauss-Kruger zone 6
EPSG	20007	Pulkovo 1995 / Gauss-Kruger zone 7
EPSG	20008	Pulkovo 1995 / Gauss-Kruger zone 8
EPSG	20009	Pulkovo 1995 / Gauss-Kruger zone 9
EPSG	2001	Antigua 1943 / British West Indies Grid
EPSG	20010	Pulkovo 1995 / Gauss-Kruger zone 10
EPSG	20011	Pulkovo 1995 / Gauss-Kruger zone 11

❏❏

[postgis_srs_codes](#), [postgis_srs](#), [postgis_srs_search](#)

7.7.9 **postgis_srs_search**

`postgis_srs_search` — Return metadata records for projected coordinate systems that have areas of usage that fully contain the bounds parameter.

Synopsis

setof record **postgis_srs_search**(geometry bounds, text auth_name=EPSG);

❏❏

Return a set of metadata records for projected coordinate systems that have areas of usage that fully contain the bounds parameter. Each record will have the `auth_name`, `auth_srid`, `sname`, `srtext`, `proj4text`, and the corners of the area of usage, `point_sw` and `point_ne`.

The search only looks for projected coordinate systems, and is intended for users to explore the possible systems that work for the extent of their data.

Availability: 3.4.0

Proj version 6+

❏❏

Search for projected coordinate systems in Louisiana.

```
SELECT auth_name, auth_srid, sname,
       ST_AsText(point_sw) AS point_sw,
       ST_AsText(point_ne) AS point_ne
FROM postgis_srs_search('SRID=4326;LINESTRING(-90 30, -91 31)')
LIMIT 3;
```


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

GEOS 版本

1.1.0 版本。

[ST_BuildArea](#), [ST_BdMPolyFromText](#)

7.8.1.2 ST_BdMPolyFromText

`ST_BdMPolyFromText` — 将 WKT 多边形文本转换为多边形几何体。

Synopsis

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

WKT 多边形文本。



Note

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

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

GEOS 版本

1.1.0 版本。

[ST_BuildArea](#), [ST_BdPolyFromText](#)

7.8.1.3 ST_GeogFromText

`ST_GeogFromText` — 将 WKT (GEOGRAPHY) 文本转换为地理几何体。

Synopsis

geography **ST_GeogFromText**(text EWKT);

例

WKT 格式 WKT 格式。 SRID 4326 格式。 ST_GeographyFromText 格式。

例

```
--- converting lon lat coords to geography
ALTER TABLE sometable ADD COLUMN geog geography(POINT,4326);
UPDATE sometable SET geog = ST_GeogFromText('SRID=4326;POINT(' || lon || ' ' || lat || ')') ←
;

--- specify a geography point using EPSG:4267, NAD27
SELECT ST_AsEWKT(ST_GeogFromText('SRID=4267;POINT(-77.0092 38.889588)'));
```

例

ST_AsText, ST_GeographyFromText

7.8.1.4 ST_GeographyFromText

ST_GeographyFromText — WKT (例) 格式。

Synopsis

geography **ST_GeographyFromText**(text EWKT);

例

WKT 格式。 SRID 4326 格式。

例

ST_GeogFromText, ST_AsText

7.8.1.5 ST_GeomCollFromText

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

Synopsis

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

几何

Makes a collection Geometry from the Well-Known-Text (WKT) representation with the given SRID. If SRID is not given, it defaults to 0.

OGC 3.2.6.2 - 符合 SRID (conformance suite) 。

WKT (GEOMETRYCOLLECTION) null 。



Note

WKT 符合 , 符合 。 符合 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.

几何

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

几何

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

7.8.1.6 ST_GeomFromEWKT

ST_GeomFromEWKT — EWKT(Extended Well-Known Text) 几何 ST_Geometry 。

Synopsis

geometry **ST_GeomFromEWKT**(text EWKT);

几何

OGC EWKT(Extended Well-Known Text) 几何 PostgreSQL ST_Geometry 。



Note

EWKT 符合 OGC 符合, SRID(符合) 符合 PostgreSQL 符合 。

符合: 2.0.0 符合 (polyhedral surface) 符合 TIN 符合。



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



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



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

❏

```
SELECT ST_GeomFromEWKT('SRID=4269;LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)');
SELECT ST_GeomFromEWKT('SRID=4269;MULTILINESTRING((-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932))');

SELECT ST_GeomFromEWKT('SRID=4269;POINT(-71.064544 42.28787)');

SELECT ST_GeomFromEWKT('SRID=4269;POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 42.3903701743239,-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 42.3902909739571)))');

SELECT ST_GeomFromEWKT('SRID=4269;MULTIPOLYGON((( -71.1031880899493 42.3152774590236,
-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,
-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,
-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,
-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,
-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,
-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,
-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,
-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,
-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,
-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,
-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,
-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,
-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,
-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,
-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,
-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,
-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,
-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,
-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,
-71.1031880899493 42.3152774590236))),
((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873 42.315113108546)))');
```

```
--3d circular string
SELECT ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)');
```

```
--Polyhedral Surface example
SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE(
  ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)');
```

❏

ST_AsEWKT, ST_GeomFromText

7.8.1.7 ST_GeomFromMARC21

ST_GeomFromMARC21 — Takes MARC21/XML geographic data as input and returns a PostGIS geometry object.

Synopsis

geometry **ST_GeomFromMARC21** (text marcxml);

国国

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.

国国

Converting MARC21/XML geographic data containing a single POINT encoded as hddd.dddddd

```
SELECT
  ST_AsText(
    ST_GeomFromMARC21('
      <record xmlns="http://www.loc.gov/MARC21/slim">
        <leader
>00000nz a2200000nc 4500</leader>
        <controlfield tag="001"
>040277569</controlfield>
          <datafield tag="034" ind1=" " ind2=" ">
            <subfield code="d"
>W004.500000</subfield>
              <subfield code="e"
>W004.500000</subfield>
                <subfield code="f"
>N054.250000</subfield>
                  <subfield code="g"
```



```

>N054.250000</subfield>
                                </datafield>
                                </record>
>' ));

      st_astext
-----
POINT(-4.5 54.25)
(1 row)

```

Converting MARC21/XML geographic data containing a single POLYGON encoded as hddmmss

```

      SELECT
      ST_AsText(
        ST_GeomFromMARC21('
          <record xmlns="http://www.loc.gov/MARC21/slim">
            <leader
>01062cem a2200241 a 4500</leader>
            <controlfield tag="001"
> 84696781 </controlfield>
              <datafield tag="034" ind1="1" ind2=" ">
                <subfield code="a"
>a</subfield>
                <subfield code="b"
>50000</subfield>
                <subfield code="d"
>E0130600</subfield>
                <subfield code="e"
>E0133100</subfield>
                <subfield code="f"
>N0523900</subfield>
                <subfield code="g"
>N0522300</subfield>
              </datafield>
            </record>
          </>
        )
      )
      st_astext
-----
POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667  ←
52.38333333333333,13.1 52.38333333333333,13.1 52.65))
(1 row)

```

Converting MARC21/XML geographic data containing a POLYGON and a POINT:

```

      SELECT
      ST_AsText(
        ST_GeomFromMARC21('
          <record xmlns="http://www.loc.gov/MARC21/slim">
            <datafield tag="034" ind1="1" ind2=" ">
              <subfield code="a"
>a</subfield>
              <subfield code="b"
>50000</subfield>
              <subfield code="d"

```

```

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

```

st_astext ↩

```

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

```

☒☒

ST_AsMARC21

7.8.1.8 ST_GeometryFromText

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

Synopsis

```

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

```

☒☒



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



This method implements the SQL/MM specification. SQL-MM 3: 5.1.40

☒☒

ST_GeomFromText

7.8.1.9 ST_GeomFromText

ST_GeomFromText — WKT 字符串 ST_Geometry 类型。

Synopsis

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

注意

OGC WKT(Well-Known Text) 字符串 PostGIS ST_Geometry 类型。



Note

ST_GeomFromText 函数接受 2 个参数，第一个是 SRID 值。如果 SRID 为 0 (SRID=0) 则返回空值。如果 SRID 不为 0 则返回 SRID 值。如果 SRID 不为 0 则返回 SRID 值。

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). 3.2.6.2 - SRID 值 (conformance suite) 值。
- ✔ 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

注意: PostGIS 2.0.0 版本中 ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') 返回空值。PostGIS 2.0.0 版本中, SQL/MM 标准中 ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') 返回空值。

注意

```
SELECT ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)');
SELECT ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)',4269);

SELECT ST_GeomFromText('MULTILINESTRING((-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932))');

SELECT ST_GeomFromText('POINT(-71.064544 42.28787)');

SELECT ST_GeomFromText('POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 42.3903701743239,
```

```

-71.1776063012595 42.3903825660754, -71.1775826583081 42.3903033653531, -71.1776585052917 42.3902909739571))');

SELECT ST_GeomFromText('MULTIPOLYGON((( -71.1031880899493 42.3152774590236,
-71.1031627617667 42.3152960829043, -71.102923838298 42.3149156848307,
-71.1023097974109 42.3151969047397, -71.1019285062273 42.3147384934248,
-71.102505233663 42.3144722937587, -71.10277487471 42.3141658254797,
-71.103113945163 42.3142739188902, -71.10324876416 42.31402489987,
-71.1033002961013 42.3140393340215, -71.1033488797549 42.3139495090772,
-71.103396240451 42.3138632439557, -71.1041521907712 42.3141153348029,
-71.1041411411543 42.3141545014533, -71.1041287795912 42.3142114839058,
-71.1041188134329 42.3142693656241, -71.1041112482575 42.3143272556118,
-71.1041072845732 42.3143851580048, -71.1041057218871 42.3144430686681,
-71.1041065602059 42.3145009876017, -71.1041097995362 42.3145589148055,
-71.1041166403905 42.3146168544148, -71.1041258822717 42.3146748022936,
-71.1041375307579 42.3147318674446, -71.1041492906949 42.3147711126569,
-71.1041598612795 42.314808571739, -71.1042515013869 42.3151287620809,
-71.1041173835118 42.3150739481917, -71.1040809891419 42.3151344119048,
-71.1040438678912 42.3151191367447, -71.1040194562988 42.3151832057859,
-71.1038734225584 42.3151140942995, -71.1038446938243 42.3151006300338,
-71.1038315271889 42.315094347535, -71.1037393329282 42.315054824985,
-71.1035447555574 42.3152608696313, -71.1033436658644 42.3151648370544,
-71.1032580383161 42.3152269126061, -71.103223066939 42.3152517403219,
-71.1031880899493 42.3152774590236))),
((-71.1043632495873 42.315113108546, -71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829, -71.1043850704575 42.3150793250568, -71.1043632495873 42.315113108546))))', 4326);

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

```


ST_GeomFromEWKT, ST_GeomFromWKB, ST_SRID

7.8.1.10 ST_LineFromText

ST_LineFromText — `SRID` `WKT` `SRID` `0`.

Synopsis

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



Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. If WKT passed in is not a LINESTRING, then null is returned.



Note

OGC 3.2.6.2 - SRID (conformance suite).

**Note**

ST_GeomFromText 函数接受 WKT 字符串并返回几何对象。ST_GeomFromText 函数是 ST_GeomFromText 函数的别名。



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



This method implements the SQL/MM specification. SQL-MM 3: 7.2.8

SQL

```
SELECT ST_LineFromText('LINESTRING(1 2, 3 4)') AS aline, ST_LineFromText('POINT(1 2)') AS
null_return;
aline | null_return
-----|-----
01020000000200000000000000000000F ... | t
```

SQL

ST_GeomFromText

7.8.1.11 ST_MLineFromText

ST_MLineFromText — WKT 字符串转换为 ST_MultiLineString 几何对象。

Synopsis

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

SQL

Makes a Geometry from Well-Known-Text (WKT) with the given SRID. If SRID is not given, it defaults to 0.

OGC 3.2.6.2 - SRID 参数 (conformance suite) 是可选的。

WKT 字符串为 null 时返回 null。

**Note**

WKT 字符串必须有效，否则将返回 null。ST_GeomFromText 函数是 ST_GeomFromText 函数的别名。



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



This method implements the SQL/MM specification. SQL-MM 3: 9.4.4

例

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

例

ST_GeomFromText

7.8.1.12 ST_MPointFromText

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

Synopsis

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

例

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

OGC 3.2.6.2 - SRID (conformance suite).

WKT null.



Note

WKT, 4326. ST_GeomFromText.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1.3.2.6.2](#)



This method implements the SQL/MM specification. SQL-MM 3: 9.2.4

例

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

例

ST_GeomFromText

7.8.1.13 ST_MPolyFromText

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

Synopsis

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

几何

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

OGC 3.2.6.2 - 几何 SRID 符合性套件 符合性套件.

WKT 符合性套件.



Note

WKT 符合性套件, 符合性套件. 符合性套件
 符合性套件 ST_GeomFromText 符合性套件.



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



This method implements the SQL/MM specification. SQL-MM 3: 9.6.4

几何

```
SELECT ST_MPolyFromText('MULTIPOLYGON(((0 0 1,20 0 1,20 20 1,0 20 1,0 0 1),(5 5 3,5 7 3,7 7 3,7 5 3,5 5 3)))');
SELECT ST_MPolyFromText('MULTIPOLYGON((( -70.916 42.1002, -70.9468 42.0946, -70.9765 42.0872, -70.9754 42.0875, -70.9749 42.0879, -70.9752 42.0881, -70.9754 42.0891, -70.9758 42.0894, -70.9759 42.0897, -70.9759 42.0899, -70.9754 42.0902, -70.9756 42.0906, -70.9753 42.0907, -70.9753 42.0917, -70.9757 42.0924, -70.9755 42.0928, -70.9755 42.0942, -70.9751 42.0948, -70.9755 42.0953, -70.9751 42.0958, -70.9751 42.0962, -70.9759 42.0983, -70.9767 42.0987, -70.9768 42.0991, -70.9771 42.0997, -70.9771 42.1003, -70.9768 42.1005, -70.977 42.1011, -70.9766 42.1019, -70.9768 42.1026, -70.9769 42.1033, -70.9775 42.1042, -70.9773 42.1043, -70.9776 42.1043, -70.9778 42.1048, -70.9773 42.1058, -70.9774 42.1061, -70.9779 42.1065, -70.9782 42.1078, -70.9788 42.1085, -70.9798 42.1087, -70.9806 42.109, -70.9807 42.1093, -70.9806 42.1099, -70.9809 42.1109, -70.9808 42.1112, -70.9798 42.1116, -70.9792 42.1127, -70.979 42.1129, -70.9787 42.1134, -70.979 42.1139, -70.9791 42.1141, -70.9987 42.1116, -71.0022 42.1273, -70.9408 42.1513, -70.9315 42.1165, -70.916 42.1002)))',4326);
```

几何

ST_GeomFromText, ST_SRID

7.8.1.14 ST_PointFromText

ST_PointFromText — 几何 SRID 几何 WKT 符合性套件. SRID 符合性套件, 符合性套件 0 符合性套件.

Synopsis

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

函数

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, 第一个是 SRID 值, 第二个是 WKT 字符串。如果 SRID 未指定, 则默认为 0。如果 WKT 字符串不是有效的点表示, 则返回 null。如果 WKT 字符串完全无效, 则抛出错误。



Note

WKT 字符串必须遵循 OGC 规范。如果 WKT 字符串不是有效的点表示, 则返回 null。如果 WKT 字符串完全无效, 则抛出错误。此函数与 `ST_GeomFromText` 类似, 但只接受点表示。OGC 规范中, `ST_MakePoint` 是 OGC 规范中的 `ST_Point` 函数。



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



This method implements the SQL/MM specification. SQL-MM 3: 6.1.8

函数

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)');
SELECT ST_PointFromText('POINT(-71.064544 42.28787)', 4326);
```

函数

[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);
```

函数

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 - 指定 SRID 值 (conformance suite) 部分。

**Note**

WKT 字符串格式, 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

例

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

```
SELECT ST_PolygonFromText('POINT(1 2)') IS NULL as point_is_notpoly;
point_is_not_poly
-----
t
```

例

ST_GeomFromText

7.8.1.16 ST_WKTToSQL

`ST_WKTToSQL` — WKT(Well-Known Text) 字符串格式 ST_Geometry 字符串。使用 `ST_GeomFromText` 函数。

Synopsis

geometry **ST_WKTToSQL**(text WKT);

例



This method implements the SQL/MM specification. SQL-MM 3: 5.1.34

例

ST_GeomFromText

7.8.2 Well-Known Binary (WKB)

7.8.2.1 ST_GeogFromWKB

ST_GeogFromWKB — WKB 转 EWKB(转 WKB) 转地理几何。

Synopsis

geography **ST_GeogFromWKB**(bytea wkb);

注意

ST_GeogFromWKB 转 WKB 转 PostGIS 转 WKB 转地理几何。 转 SQL 转 (Geometry Factory) 转。

SRID 转, 转 4326(WGS84 转) 转。



This method supports Circular Strings and Curves.

注意

```
--Although bytea rep contains single \, these need to be escaped when inserting into a table
SELECT ST_AsText(
ST_GeogFromWKB(E'\\001\\002\\000\\000\\000\\002\\000\\000\\000\\037\\205\\353Q
\\270~\\\\\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\\\\\300)\\\\\\217\\302\\365\\230
C@')
);
--
-- st_astext
--
LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)
```

注意

ST_GeogFromText, **ST_AsBinary**

7.8.2.2 ST_GeomFromEWKB

ST_GeomFromEWKB — EWKB(Extended Well-Known Binary) 转 ST_Geometry 转。

Synopsis

geometry **ST_GeomFromEWKB**(bytea EWKB);

函数

OGC EWKB(Extended Well-Known Binary) 函数返回 PostGIS ST_Geometry 类型。



Note

EWKB 函数 OGC 规范, SRID(空间参考系统ID) 返回 PostGIS 类型。

函数: 2.0.0 函数 (polyhedral surface) 和 TIN 函数。



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).

函数

NAD83 函数 (SRID 4269) 函数 LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932) 函数



Note

函数: 函数 (\\) 函数 (') 函数, standard_conforming_strings 函数 \ 函数. 函数 AsEWKB 函数.

```
SELECT ST_GeomFromEWKB(E'\\001\\002\\000\\000 \\255\\020\\000\\000\\003\\000\\000\\000\\344 ←
J=
\\013B\\312Q\\300n\\303(\\010\\036!E@' '\\277E' 'K
\\312Q\\300\\366{b\\235*!E@\\225|\\354.P\\312Q
\\300p\\231\\323e1!E@');
```



Note

In PostgreSQL 9.1+ - standard_conforming_strings is set to on by default, where as in past versions it was set to off. You can change defaults as needed for a single query or at the database or server level. Below is how you would do it with standard_conforming_strings = on. In this case we escape the ' with standard ansi ', but slashes are not escaped

```
set standard_conforming_strings = on;
SELECT ST_GeomFromEWKB('\\001\\002\\000\\000 \\255\\020\\000\\000\\003\\000\\000\\000\\344J=\\012\\013B
\\312Q\\300n\\303(\\010\\036!E@' '\\277E' 'K\\012\\312Q\\300\\366{b\\235*!E@\\225|\\354.P\\312Q\\012\\300 ←
p\\231\\323e1')
```

函数

ST_AsBinary, ST_AsEWKB, ST_GeomFromWKB

7.8.2.3 ST_GeomFromWKB

ST_GeomFromWKB — WKB(Well-Known Binary) 转 SRID 的函数。

Synopsis

```
geometry ST_GeomFromWKB(bytea geom);
geometry ST_GeomFromWKB(bytea geom, integer srid);
```

描述

ST_GeomFromWKB 将 WKB 转 SRID(指定 ID) 的函数。该函数是 SQL (Geometry Factory) 的一部分。该函数 ST_WKBToSQL 是 ST_GeomFromWKB 的逆函数。

SRID 指定, 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.

描述

```
--Although bytea rep contains single \, these need to be escaped when inserting into a table
-- unless standard_conforming_strings is set to on.
SELECT ST_AsEWKT(
ST_GeomFromWKB(E'\\001\\002\\000\\000\\000\\002\\000\\000\\000\\037\\205\\353Q
\\270~\\111\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\111\\300)\\111\\217\\302\\365\\230
C@',4326)
);
          st_asewkt
-----
SRID=4326;LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)

SELECT
  ST_AsText(
    ST_GeomFromWKB(
      ST_AsEWKB('POINT(2 5)::geometry)
    )
  );
  st_astext
-----
POINT(2 5)
(1 row)
```

描述

[ST_WKBToSQL](#), [ST_AsBinary](#), [ST_GeomFromEWKB](#)

7.8.2.4 ST_LineFromWKB

ST_LineFromWKB — 给定 SRID 的 WKB 数据返回 LINESTRING 几何。

Synopsis

```
geometry ST_LineFromWKB(bytea WKB);
geometry ST_LineFromWKB(bytea WKB, integer srid);
```

注意

ST_LineFromWKB 接受 WKB 数据，SRID(可选 ID) 和 LINESTRING 几何 - 返回 LINESTRING 几何。使用 SQL Geometry Factory 函数。SRID 默认为 0。bytea 数据类型，NULL 返回 NULL。



Note

OGC 3.2.6.2 - 符合 SRID 规范 (conformance suite)。



Note

使用 LINESTRING 几何，ST_GeomFromWKB 返回几何。使用 ST_GeomFromWKB 函数，返回几何。

✓ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1 s3.2.6.2](#)

✓ This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

注意

```
SELECT ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('LINESTRING(1 2, 3 4)'))) AS aline,
       ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('POINT(1 2)'))) IS NULL AS null_return;
aline                                     | null_return
-----|-----
01020000000200000000000000000000F ... | t
```

注意

ST_GeomFromWKB, ST_LinestringFromWKB

7.8.2.5 ST_LinestringFromWKB

ST_LinestringFromWKB — 给定 SRID 的 WKB 数据返回 LINESTRING 几何。

Synopsis

geometry **ST_GeomFromWKB**(bytea geom);
 geometry **ST_GeomFromWKB**(bytea geom, integer srid);

描述

ST_PointFromWKB 将 WKB 格式 SRID(空间参考系统 ID) 转换为 POINT 类型 - 点。 使用 SQL 几何工厂 (Geometry Factory) 解析。

SRID 默认为 0。 如果 geom 为 NULL，则返回 NULL。

- ✓ 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.

示例

```
SELECT
  ST_AsText(
    ST_PointFromWKB(
      ST_AsEWKB('POINT(2 5)::geometry')
    )
  );
 st_astext
-----
POINT(2 5)
(1 row)

SELECT
  ST_AsText(
    ST_PointFromWKB(
      ST_AsEWKB('LINESTRING(2 5, 2 6)::geometry')
    )
  );
 st_astext
-----
(1 row)
```

参见

[ST_GeomFromWKB](#), [ST_LineFromWKB](#)

7.8.2.7 ST_WKBToSQL

ST_WKBToSQL — WKB(Well-Known Binary) 格式 ST_Geometry 转换为 SQL 格式。 使用 SRID 默认为 0。 如果 geom 为 NULL，则返回 NULL。

Synopsis

geometry **ST_WKBToSQL**(bytea WKB);

⚠

 This method implements the SQL/MM specification. SQL-MM 3: 5.1.36

⚠

ST_GeomFromWKB

7.8.3 Other Formats

7.8.3.1 ST_Box2dFromGeoHash

ST_Box2dFromGeoHash — GeoHash 字符串 BOX2D 值。

Synopsis

box2d **ST_Box2dFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

⚠

GeoHash 字符串 BOX2D 值。

If no precision is specified ST_Box2dFromGeoHash returns a BOX2D based on full precision of the input GeoHash string.

precision 值, ST_Box2dFromGeoHash 将 GeoHash 字符串转换为 BOX2D 值。返回的 BOX2D 值将包含 GeoHash 字符串所表示的整个区域。

2.1.0 版本引入。

⚠

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0');
```

```
          st_geomfromgeohash
```

```
-----
BOX(-115.172816 36.114646,-115.172816 36.114646)
```

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 0);
```

```
          st_box2dfromgeohash
```

```
-----
BOX(-180 -90,180 90)
```

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10);
```

```
          st_box2dfromgeohash
```

```
-----
BOX(-115.17282128334 36.1146408319473,-115.172810554504 36.1146461963654)
```


函数

[ST_GeoHash](#), [ST_GeomFromGeoHash](#), [ST_PointFromGeoHash](#)

7.8.3.2 ST_GeomFromGeoHash

ST_GeomFromGeoHash — GeoHash 字符串转换为几何对象。

Synopsis

geometry **ST_GeomFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

函数

GeoHash 字符串转换为几何对象。精度由 GeoHash 字符串的精度决定。

precision 指定精度，ST_GeomFromGeoHash 使用 GeoHash 字符串的精度。

precision 指定精度，ST_GeomFromGeoHash 使用 GeoHash 字符串的精度。

2.1.0 版本引入。

函数

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0'));
           st_astext
-----
POLYGON((-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646))

SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 4));
           st_astext
-----
POLYGON((-115.3125 36.03515625,-115.3125 36.2109375,-114.9609375 36.2109375,-114.9609375 36.03515625,-115.3125 36.03515625))

SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10));
           st_astext
-----
POLYGON((-115.17282128334 36.1146408319473,-115.17282128334 36.1146461963654,-115.172810554504 36.1146461963654,-115.172810554504 36.1146408319473,-115.17282128334 36.1146408319473))
```

函数

[ST_GeoHash](#), [ST_Box2dFromGeoHash](#), [ST_PointFromGeoHash](#)

7.8.3.3 ST_GeomFromGML

ST_GeomFromGML — 将 GML 几何对象转换为 PostGIS 几何对象。

Synopsis

```
geometry ST_GeomFromGML(text geomgml);
geometry ST_GeomFromGML(text geomgml, integer srid);
```

返回

OGC GML 几何对象转换为 PostGIS ST_Geometry 几何对象。

ST_GeomFromGML 将 GML 几何对象 (geometry fragment) 转换为 PostGIS 几何对象。该 GML 几何对象必须是有效的。

支持的 OGC GML 几何对象类型:

- GML 3.2.1 几何对象
- GML 3.1.1 几何对象 SF-2 (GML 3.1.0 到 3.0.0 几何对象)
- GML 2.1.2

OGC GML 规范: <http://www.opengeospatial.org/standards/gml>

1.5 版本使用 LibXML2 1.6 版本。

支持的: 2.0.0 版本 (polyhedral surface) 和 TIN 几何对象。

支持的: 2.0.0 版本 SRID 几何对象。



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

GML 几何对象 (MultiGeometry) 支持 2D 和 3D 几何对象。PostGIS 支持 Z 索引的 ST_GeomFromGML 函数将 2D 几何对象转换为 3D 几何对象。

GML 几何对象支持 SRS 索引。PostGIS 支持 SRS 索引, 使用 ST_GeomFromGML 函数将 SRS 索引的 GML 几何对象转换为 PostGIS 几何对象, 使用 srsName 属性指定 SRS 名称。

ST_GeomFromGML 函数将 GML 几何对象转换为 PostGIS 几何对象。该 GML 几何对象必须是有效的。该 GML 几何对象支持 XLink 属性。



Note

ST_GeomFromGML 函数支持 SQL/MM 几何对象。

例: srsName 指定

```
SELECT ST_GeomFromGML($$
  <gml:LineString xmlns:gml="http://www.opengis.net/gml"
    srsName="EPSG:4269">
    <gml:coordinates>
      -71.16028,42.258729 -71.160837,42.259112 -71.161143,42.25932
    </gml:coordinates>
  </gml:LineString>
$$);
```

例: XLink 指定

```
SELECT ST_GeomFromGML($$
  <gml:LineString xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    srsName="urn:ogc:def:crs:EPSG::4269">
    <gml:pointProperty>
      <gml:Point gml:id="p1"
        ><gml:pos
        >42.258729 -71.16028</gml:pos
        ></gml:Point>
      </gml:pointProperty>
      <gml:pos
        >42.259112 -71.160837</gml:pos>
      <gml:pointProperty>
        <gml:Point xlink:type="simple" xlink:href="#p1"/>
      </gml:pointProperty>
    </gml:LineString>
  </gml:LineString>
$$);
```

例: 多面体

```
SELECT ST_AsEWKT(ST_GeomFromGML('
  <gml:PolyhedralSurface xmlns:gml="http://www.opengis.net/gml">
  <gml:polygonPatches>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing
        ><gml:posList srsDimension="3"
        >0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList
        ></gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
      <gml:PolygonPatch>
        <gml:exterior>
          <gml:LinearRing
          ><gml:posList srsDimension="3"
          >0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList
          ></gml:LinearRing>
          </gml:exterior>
        </gml:PolygonPatch>
        <gml:PolygonPatch>
          <gml:exterior>
            <gml:LinearRing
            ><gml:posList srsDimension="3"
```

```

>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList
></gml:LinearRing>
  </gml:exterior>
</gml:PolygonPatch>
<gml:PolygonPatch>
  <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList
></gml:LinearRing>
  </gml:exterior>
</gml:PolygonPatch>
<gml:PolygonPatch>
  <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 0 0 1 0</gml:posList
></gml:LinearRing>
  </gml:exterior>
</gml:PolygonPatch>
<gml:PolygonPatch>
  <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList
></gml:LinearRing>
  </gml:exterior>
</gml:PolygonPatch>
</gml:polygons>
</gml:PolyhedralSurface
>'));

-- result --
POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))

```

☒☒

Section [2.2.3](#), [ST_AsGML](#), [ST_GMLToSQL](#)

7.8.3.4 ST_GeomFromGeoJSON

ST_GeomFromGeoJSON — GeoJSON 格式数据转换为 PostGIS 几何类型。

Synopsis

```

geometry ST_GeomFromGeoJSON(text geomjson);
geometry ST_GeomFromGeoJSON(json geomjson);
geometry ST_GeomFromGeoJSON(jsonb geomjson);

```

函数

GeoJSON 函数在 PostGIS 3.6.0 中引入。

ST_GeomFromGML 从 JSON 片段 (geometry fragment) 创建几何。该 JSON 片段可以是 GeoJSON 或 GML。

Enhanced: 3.0.0 解析几何默认到 SRID=4326 如果未指定其他方式。

Enhanced: 2.5.0 现在可以接受 json 和 jsonb 作为输入。

2.0.0 支持 GeoJSON 0.9。JSON-C 0.9 支持 GeoJSON 0.9。



Note

JSON-C 支持 GeoJSON 0.9，但不支持 GeoJSON 1.0。JSON-C 支持 GeoJSON 1.0，但需要指定 "--with-jsondir=/path/to/json-c" 选项。有关 Section 2.2.3 的更多信息。



This function supports 3d and will not drop the z-index.

函数

```
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[-48.23456,20.12345]}')) ←
      As wkt;
wkt
-----
POINT(-48.23456 20.12345)
```

```
-- a 3D linestring
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"LineString","coordinates": ←
      ":[1,2,3],[4,5,6],[7,8,9]}')) As wkt;
wkt
-----
LINESTRING(1 2,4 5,7 8)
```

函数

ST_AsText, ST_AsGeoJSON, Section 2.2.3

7.8.3.5 ST_GeomFromKML

ST_GeomFromKML — 从 KML 创建几何 PostGIS 3.6.0 中引入。

Synopsis

geometry **ST_GeomFromKML**(text geomkml);

函数

OGC KML 函数支持 PostGIS ST_Geometry 函数。

ST_GeomFromKML 从 KML 几何片段 (geometry fragment) 创建几何。KML 几何片段是 KML 文档的一部分。

支持的 OGC KML 版本如下：

- KML 2.2.0 版本

OGC KML 规范: <http://www.opengeospatial.org/standards/kml>

Availability: 1.5, requires libxml2 2.6+



This function supports 3d and will not drop the z-index.



Note

ST_GeomFromKML 函数 SQL/MM 兼容。

参数: **srsName** 字符串

```
SELECT ST_GeomFromKML($$
  <LineString>
    <coordinates>
  >-71.1663,42.2614
    -71.1667,42.2616</coordinates>
  </LineString>
$$);
```

函数

Section 2.2.3, ST_AsKML

7.8.3.6 ST_GeomFromTWKB

ST_GeomFromTWKB — TWKB ("Tiny Well-Known Binary") 二进制几何。

Synopsis

geometry **ST_GeomFromTWKB**(bytea twkb);

函数

ST_GeomFromTWKB 从 TWKB ("Tiny Well-Known Binary") 二进制 (WKB) 几何创建几何。


```
SELECT ST_AsText(ST_GeomFromTWKB(ST_AsTWKB('LINESTRING(126 34, 127 35)')::geometry));
```

```

      st_astext
-----
LINESTRING(126 34, 127 35)
(1 row)

```

```
SELECT ST_AsEWKT(
  ST_GeomFromTWKB(E'\x620002f7f40dbce4040105')
);
```

st_asewkt
LINESTRING(-113.98 39.198,-113.981 39.195)

(1 row)

ST AsTWKB

7.8.3.7 ST_GMLToSQL

ST_GMLToSQL — GML `[[[...]]]` ST_Geometry `[[...]]`. `[[...]]` ST_GeomFromGML `[[...]]` `[[...]]`.

Synopsis

```
geometry ST_GMLToSQL(text geomgml);
geometry ST_GMLToSQL(text geomgml, integer srid);
```


 This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (XXXXXX)
1.5 XXXXX. LibXML2 1.6 XXXXX.

Version: 2.0.0 (polyhedral surface) TIN

Version: 2.0.0 SRID 0.



Section 2.2.3, ST_GeomFromGML, ST_AsGML

7.8.3.8 ST_LineFromEncodedPolyline

ST LineFromEncodedPolyline — (polyline) .

Synopsis

```
geometry ST_LineFromEncodedPolyline(text polyline, integer precision=5);
```

函数

函数返回一个 `LINESTRING` 类型的几何对象。

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。

函数

```
-- Create a line string from a polyline
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@'));
-- result --
SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)

-- Select different precision that was used for polyline encoding
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@',6));
-- result --
SRID=4326;LINESTRING(-12.02 3.85,-12.095 4.07,-12.6453 4.3252)
```

函数

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);

函数

GeoHash 字符串转换为点。返回一个 `POINT` 类型的几何对象。

precision 指定要保留的小数位数，`ST_PointFromGeoHash` 使用与 `GeoHash` 字符串相同的精度。

precision 指定要保留的小数位数，`ST_PointFromGeoHash` 使用与 `GeoHash` 字符串相同的精度。

2.1.0 版本引入。

函数


```

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0'));
          st_astext
-----
POINT(-115.172816 36.114646)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 4));
          st_astext
-----
POINT(-115.13671875 36.123046875)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10));
          st_astext
-----
POINT(-115.172815918922 36.1146435141563)

```

☒☒

`ST_GeoHash`, `ST_Box2dFromGeoHash`, `ST_GeomFromGeoHash`

7.8.3.10 `ST_FromFlatGeobufToTable`

`ST_FromFlatGeobufToTable` — Creates a table based on the structure of FlatGeobuf data.

Synopsis

`void` **`ST_FromFlatGeobufToTable`**(text schemaname, text tablename, bytea FlatGeobuf input data);

☒☒

Creates a table based on the structure of FlatGeobuf data. (<http://flatgeobuf.org>).

`schema` Schema name.

`table` Table name.

`data` Input FlatGeobuf data.

Availability: 3.2.0

7.8.3.11 `ST_FromFlatGeobuf`

`ST_FromFlatGeobuf` — Reads FlatGeobuf data.

Synopsis

setof anyelement **`ST_FromFlatGeobuf`**(anyelement Table reference, bytea FlatGeobuf input data);

☒☒

Reads FlatGeobuf data (<http://flatgeobuf.org>). NOTE: PostgreSQL bytea cannot exceed 1GB.

`tabletype` reference to a table type.

`data` input FlatGeobuf data.

Availability: 3.2.0

7.9 Geometry Output

7.9.1 Well-Known Text (WKT)

7.9.1.1 ST_AsEWKT

ST_AsEWKT — 返回 WKT(Well-Known Text) 格式的 SRID 标识的几何体。

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);
```

返回

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.

版本: 2.0.0 引入, 2.1.0 改进, 2.2.0 引入 TIN 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).

例

```
SELECT ST_AsEWKT('0103000020E61000000100000005000000000000
0000000000000000000000000000000000000000000000000000
F03F000000000000F03F000000000000F03F000000000000F03
F000000000000000000000000000000000000000000000000000'::geometry);

      st_asewkt
-----
SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
(1 row)

SELECT ST_AsEWKT('010800008003000000000000000060 ↵
E30A4100000000785C0241000000000000F03F0000000018
E20A4100000000485F024100000000000000400000000018
E20A4100000000305C024100000000000000840')

--st_asewkt--
CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)
```

例

[ST_AsBinary](#), [ST_AsEWKB](#), [ST_AsText](#), [ST_GeomFromEWKT](#)

7.9.1.2 ST_AsText

[ST_AsText](#) — 将 WKT(Well-Known Text) 格式转换为 SRID 格式。

Synopsis

```
text ST_AsText(geometry g1);
text ST_AsText(geometry g1, integer maxdecimaldigits = 15);
text ST_AsText(geography g1);
text ST_AsText(geography g1, integer maxdecimaldigits = 15);
```

例

Returns the OGC [Well-Known Text](#) (WKT) representation of the geometry/geography. The optional *maxdecimaldigits* argument may be used to limit the number of digits after the decimal point in output ordinates (defaults to 15).

To perform the inverse conversion of WKT representation to PostGIS geometry use [ST_GeomFromText](#).



Note

The standard OGC WKT representation does not include the SRID. To include the SRID as part of the output representation, use the non-standard PostGIS function [ST_AsEWKT](#)



Warning

The textual representation of numbers in WKT may not maintain full floating-point precision. To ensure full accuracy for data storage or transport it is best to use [Well-Known Binary](#) (WKB) format (see [ST_AsBinary](#) and *maxdecimaldigits*).



Warning

Using the `maxdecimaldigits` parameter can cause output geometry to become invalid. To avoid this use `ST_ReducePrecision` with a suitable gridsize first.

1.5.0 ☒☒☒☒☒☒☒☒☒☒☒☒.

Enhanced: 2.5 - optional parameter precision introduced.



✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
s2.1.1.1



✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.25



✔ This method supports Circular Strings and Curves.


```
SELECT ST_AsText('010300000001000000050000000000000000  
000000000000000000000000000000000000000000000000  
F03F000000000000F03F000000000000F03F000000000000F03  
F00000000000000000000000000000000000000000000000');
```

```
st_astext
-----
POLYGON((0 0,0 1,1 1,1 0,0 0))
```

Full precision output is the default.

```
SELECT ST_AsText('POINT(111.111111 1.111111)');
      st_astext
-----
POINT(111.111111 1.111111)
```

The *maxdecimaldigits* argument can be used to limit output precision.

```
SELECT ST_AsText('POINT(111.111111 1.111111)'), 2);
      st_astext
-----
POINT(111.11 1.11)
```



ST AsBinary, ST AsEWKB, ST AsEWKT, ST GeomFromText

7.9.2 Well-Known Binary (WKB)

7.9.2.1 ST AsBinary

ST_AsBinary — Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.

Synopsis

```
bytea ST_AsBinary(geometry g1);
bytea ST_AsBinary(geometry g1, text NDR_or_XDR);
bytea ST_AsBinary(geography g1);
bytea ST_AsBinary(geography g1, text NDR_or_XDR);
```

返回

Returns the OGC/ISO **Well-Known Binary** (WKB) representation of the geometry. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding: either 'NDR' for little-endian; or 'XDR' for big-endian. Supplying unknown arguments will result in little-endian output.

WKB format is useful to read geometry data from the database and maintaining full numeric precision. This avoids the precision rounding that can happen with text formats such as WKT.

To perform the inverse conversion of WKB to PostGIS geometry use **ST_GeomFromWKB**.



Note

The OGC/ISO WKB format does not include the SRID. To get the EWKB format which does include the SRID use **ST_AsEWKB**



Note

The default behavior in PostgreSQL 9.0 has been changed to output bytea in hex encoding. If your GUI tools require the old behavior, then SET bytea_output='escape' in your database.

兼容性: 2.0.0 引入, 支持 TIN 几何类型。

兼容性: 2.0.0 引入。

兼容性: 2.0.0 引入。

1.5.0 引入。

兼容性: 2.0.0 引入。在 PostgreSQL 9.0 中，`ST_AsBinary('POINT(1 2)')` 返回 `st_asbinary(unknown)` 不是唯一的错误消息。使用 `ST_AsBinary('POINT(1 2)::geometry');` 可以解决这个问题。在 `legacy.sql` 文件中。



This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**, s2.1.1.1



This method implements the SQL/MM specification. SQL-MM 3: 5.1.37



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



This function supports 3d and will not drop the z-index.

[illegible][illegible]

ST AsBinary, ST GeomFromEWKB, ST SRID

7.9.2.3 ST_AsHEXEWKB

ST AsHEXEWKB — $\square\square\square\square\square\square$ (NDR) $\square\square\square\square\square\square$ (XDR) $\square\square\square\square\square\square$ HEXEWKB ($\square\square\square$) $\square\square\square\square$
 $\square\square\square\square\square$.

Synopsis

```
text ST_AsHEXEWKB(geometry g1, text NDRorXDR);
text ST_AsHEXEWKB(geometry g1);
```



XXXXXXXXXX (NDR) XXXXXXXXX (XDR) XXXXXXXXX HEXEWKB (XXXX) XXXXXXXXXXXXXXXX. XXXXXXXX
XXXXXXXXXX NDR XXXXXXXX.



Note

1.2.2 ☒☒☒☒☒☒☒☒☒☒☒.

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.

11

```
SELECT ST_AshExEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
      which gives same answer as

SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326)::text;

st_ashexewkb
-----
0103000020E61000000100000000500
00000000000000000000000000000000
0000000000000000000000000000000F03F
000000000000F03F000000000000F03F000000000000F03
F00000000000000000000000000000000000000000000000
```

7.9.3 Other Formats

7.9.3.1 ST_AsEncodedPolyline

ST_AsEncodedPolyline — .

Synopsis

```
text ST_AsEncodedPolyline(geometry geom, integer precision=5);
```



Returns the geometry as an Encoded Polyline. This format is used by Google Maps with precision=5 and by Open Source Routing Machine with precision=5 and 6.

Optional precision specifies how many decimal places will be preserved in Encoded Polyline. Value should be the same on encoding and decoding, or coordinates will be incorrect.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.

11

```
SELECT ST_AsEncodedPolyline(GeomFromEWKT('SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)'));
-- result--
|_p~iF~ps|U ulLnnqC mqNvxq`@
```

XXXXXXXXXXXXXXXX (segmentize) XXXXXXXXXXXX, XXXXXXXXXXXX.

```
-- the SQL for Boston to San Francisco, segments every 100 KM
SELECT ST_AsEncodedPolyline(
  ST_Segmentize(
    ST_GeogFromText('LINESTRING(-71.0519 42.4935,-122.4483 37.64)'),
    100000)::geometry) As encodedFlightPath;
```

XXXX \$ XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.


```

<script type="text/javascript" src="http://maps.googleapis.com/maps/api/js?libraries= ↵
  geometry"
></script>
<script type="text/javascript">
  flightPath = new google.maps.Polyline({
    path: google.maps.geometry.encoding.decodePath("$encodedFlightPath"),
    map: map,
    strokeColor: '#0000CC',
    strokeOpacity: 1.0,
    strokeWeight: 4
  });
</script>

```

☒☒

ST_LineFromEncodedPolyline, ST_Segmentize

7.9.3.2 ST_AsFlatGeobuf

ST_AsFlatGeobuf — Return a FlatGeobuf representation of a set of rows.

Synopsis

```

bytea ST_AsFlatGeobuf(anyelement set row);
bytea ST_AsFlatGeobuf(anyelement row, bool index);
bytea ST_AsFlatGeobuf(anyelement row, bool index, text geom_name);

```

☒☒

Return a FlatGeobuf representation (<http://flatgeobuf.org>) of a set of rows corresponding to a FeatureCollection. NOTE: PostgreSQL bytea cannot exceed 1GB.

row row data with at least a geometry column.

index toggle spatial index creation. Default is false.

geom_name is the name of the geometry column in the row data. If NULL it will default to the first found geometry column.

Availability: 3.2.0

7.9.3.3 ST_AsGeobuf

ST_AsGeobuf — Return a Geobuf representation of a set of rows.

Synopsis

```

bytea ST_AsGeobuf(anyelement set row);
bytea ST_AsGeobuf(anyelement row, text geom_name);

```

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.

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;
st_asgeobuf
-----
GAAiEAo0CgwIBBoIAAAAAgIAAAE=
```

7.9.3.4 ST_AsGeoJSON

ST_AsGeoJSON — Return a geometry or feature in GeoJSON format.

Synopsis

```
text ST_AsGeoJSON(record feature, text geom_column="", integer maxdecimaldigits=9, boolean
pretty_bool=false, text id_column="");
text ST_AsGeoJSON(geometry geom, integer maxdecimaldigits=9, integer options=8);
text ST_AsGeoJSON(geography geog, integer maxdecimaldigits=9, integer options=0);
```

国国

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 (国: EPSG:4326)
- 4: GeoJSON Long CRS (国: urn:ogc:def:crs:EPSG::4326)
- 8: GeoJSON Short CRS if not EPSG:4326 (default)

The `id_column` parameter is used to set the "id" member of the returned GeoJSON features. As per GeoJSON RFC, this SHOULD be used whenever a feature has a commonly used identifier, such as a primary key. When not specified, the produced features will not get an "id" member and any columns other than the geometry, including any potential keys, will just end up inside the feature's "properties" member.

The GeoJSON specification states that polygons are oriented using the Right-Hand Rule, and some clients require this orientation. This can be ensured by using `ST_ForcePolygonCCW`. The specification also requires that geometry be in the WGS84 coordinate system (SRID = 4326). If necessary geometry can be projected into WGS84 using `ST_Transform`: `ST_Transform( geom, 4326 )`.

GeoJSON can be tested and viewed online at geojson.io and geojsonlint.com. It is widely supported by web mapping frameworks:

- [OpenLayers GeoJSON Example](#)
- [Leaflet GeoJSON Example](#)
- [Mapbox GL GeoJSON Example](#)

1.3.4 国国国国国国国国国国.

1.5.0 国国国国国国国国国国.

国国国: 2.0.0 国国国国国国国国国 (default arg) 国国国国国国国 (named arg) 国国国国国.

Changed: 3.0.0 support records as input

Changed: 3.0.0 output SRID if not EPSG:4326.

Changed: 3.5.0 allow specifying the column containing the feature id



This function supports 3d and will not drop the z-index.

国国

Generate a FeatureCollection:

```
SELECT json_build_object(
    'type', 'FeatureCollection',
    'features', json_agg(ST_AsGeoJSON(t.*, id_column =
> 'id')::json)
)
FROM ( VALUES (1, 'one', 'POINT(1 1)::geometry),
              (2, 'two', 'POINT(2 2)'),
              (3, 'three', 'POINT(3 3)')
      ) as t(id, name, geom);
```

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [1, 1]
      },
      "id": 1,
      "properties": {
        "name": "one"
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [2, 2]
      },
      "id": 2,
      "properties": {
        "name": "two"
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [3, 3]
      },
      "id": 3,
      "properties": {
        "name": "three"
      }
    }
  ]
}
```

Generate a Feature:

```
SELECT ST_AsGeoJSON(t.*, id_column =
> 'id')
FROM (VALUES (1, 'one', 'POINT(1 1)::geometry)) AS t(id, name, geom);
```

```
st_asgeojson
```

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [1, 1]
  },
  "id": 1,
  "properties": {
    "name": "one"
  }
}
```

Don't forget to transform your data to WGS84 longitude, latitude to conform with the GeoJSON specification:

```
SELECT ST_AsGeoJSON(ST_Transform(geom, 4326)) from fe_edges limit 1;
```

```
st_asgeojson
```

```
{
  "type": "MultiLineString",
  "coordinates": [
    [
      [
        [-89.734634999999997, 31.492072000000000],
        [-89.734959999999997, 31.492237999999997]
      ]
    ]
  ]
}
```

3D geometries are supported:

```
SELECT ST_AsGeoJSON('LINESTRING(1 2 3, 4 5 6)');
```

```
{
  "type": "LineString",
  "coordinates": [
    [1, 2, 3],
    [4, 5, 6]
  ]
}
```

Options argument can be used to add BBOX and CRS in GeoJSON output:

```
SELECT ST_AsGeoJSON(ST_SetSRID('POINT(1 1)::geometry', 4326), 9, 4|1);
```

```
{
  "type": "Point",
  "crs": {
    "type": "name",
    "properties": {
      "name": "urn:ogc:def:crs:EPSG::4326"
    }
  },
  "bbox": [1.000000000, 1.000000000, 1.000000000, 1.000000000],
  "coordinates": [1, 1]
}
```

☒☒

ST_GeomFromGeoJSON, **ST_ForcePolygonCCW**, **ST_Transform**

7.9.3.5 ST_AsGML

ST_AsGML — ☒☒☒ GML 2 ☒☒ GML 3 ☒☒☒☒☒☒☒☒☒☒.

Synopsis

```
text ST_AsGML(geometry geom, integer maxdecimaldigits=15, integer options=0);
text ST_AsGML(geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null,
text id=null);
text ST_AsGML(integer version, geometry geom, integer maxdecimaldigits=15, integer options=0,
text nprefix=null, text id=null);
text ST_AsGML(integer version, geography geog, integer maxdecimaldigits=15, integer options=0,
text nprefix=null, text id=null);
```

返回

Return the geometry as a Geography Markup Language (GML) element. The version parameter, if specified, may be either 2 or 3. If no version parameter is specified then the default is assumed to be 2. The *maxdecimaldigits* argument may be used to reduce the maximum number of decimal places used in output (defaults to 15).



Warning

Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use **ST_ReducePrecision** with a suitable gridsize first.

GML 2 2.1.2 函数, GML 3 3.1.1 函数。

' 参数 (bitfield) 控制 CRS 和 GML 输出选项, 如表所示。

- 0: GML Short CRS (如: EPSG:4326), 输出
- 1: GML Long CRS (如: urn:ogc:def:crs:EPSG::4326)
- 2: GML 3 输出, 输出 srsDimension 属性。
- 4: GML 3 输出, 输出 <Curve> 和 <LineString> 属性。
- 16: 输出 (如: srid=4326) 输出。 输出 (axis order) 输出, GML 3.1.1 输出。 输出, 输出。
- 32: 输出 (envelope) 输出。

输出 (输出) 输出' 输出' 输出。 输出 NULL 输出'gml' 输出。

1.3.2 函数。

1.5.0 函数。

输出: 2.0.0 输出。 输出 GML 3 输出'4' 输出。 GML 3 输出 TIN 输出。 输出'32' 输出。

输出: 2.0.0 输出 (named arg) 输出。

输出: 2.1.0 输出 GML 3 输出 ID 输出。



Note

ST_AsGML 输出 3 输出 TIN 输出。

- ✔ 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).

国国: 国国 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>
```

国国: 国国 3

```
-- Flip coordinates and output extended EPSG (16 | 1)--
SELECT ST_AsGML(3, ST_GeomFromText('POINT(5.234234233242 6.34534534534)',4326), 5, 17);
  st_asgml
-----
<gml:Point srsName="urn:ogc:def:crs:EPSG::4326"
><gml:pos
>6.34535 5.23423</gml:pos
></gml:Point>
```

```
-- Output the envelope (32) --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 32);
  st_asgml
-----
<gml:Envelope srsName="EPSG:4326">
  <gml:lowerCorner
>1 2</gml:lowerCorner>
  <gml:upperCorner
>10 20</gml:upperCorner>
</gml:Envelope>
```

```
-- Output the envelope (32) , reverse (lat lon instead of lon lat) (16), long srs (1)= 32 | ←
16 | 1 = 49 --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 49);
  st_asgml
-----
<gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
  <gml:lowerCorner
>2 1</gml:lowerCorner>
  <gml:upperCorner
```

```
>20 10</gml:upperCorner>
</gml:Envelope>
```

```
-- Polyhedral Example --
SELECT ST_AsGML(3, ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0) ←
),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )''));
st_asgml
-----
<gml:PolyhedralSurface>
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 1 0 0 1 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
```


**Note**

ST_AskML 返回 SRID 4326 的 KML 字符串。



This function supports 3d and will not drop the z-index.

例

```
SELECT ST_AskML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```

st_askml
-----
<Polygon
><outerBoundaryIs
><LinearRing
><coordinates
>0,0 0,1 1,1 1,0 0,0</coordinates
></LinearRing
></outerBoundaryIs
></Polygon>

--3d linestring
SELECT ST_AskML('SRID=4326;LINESTRING(1 2 3, 4 5 6)');
<LineString
><coordinates
>1,2,3 4,5,6</coordinates
></LineString>

```

例

[ST_AsSVG](#), [ST_AsGML](#)

7.9.3.7 ST_AsLatLonText

ST_AsLatLonText — 返回指定几何体的经纬度文本。

Synopsis

text **ST_AsLatLonText**(geometry pt, text format=“”);

例

ST_AsLatLonText(geom, 'WKT').

**Note**

ST_AsLatLonText 返回的文本格式为 "X(经度) Y(纬度)" 格式。经度范围是 -180 到 180 度，纬度范围是 -90 到 90 度。

返回的字符串格式为 "D", "M", "S", (方向, cardinal direction) "C" 格式。D, M, S 格式为 "SSS.SSSS" ("1.0023" 格式)。

M, S, C 格式。"C" 格式为 "SSS.SSSS" 格式。"S" 格式为 "SSS.SSSS" 格式。"M" 格式为 "SSS.SSSS" 格式。

返回的字符串格式为 (方向 0 度) 格式。

2.0 格式。

SQL

SQL

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)'));
      st_aslatlon
-----
2°19'29.928"S 3°14'3.243"W
```

(方向) 格式。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D°M'S.SSS"C'));
      st_aslatlon
-----
2°19'29.928"S 3°14'3.243"W
```

D, M, S, C 格式。格式。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D degrees, M minutes, S seconds to
the C'));
      st_aslatlon
-----
2 degrees, 19 minutes, 30 seconds to the S 3 degrees, 14 minutes, 3 seconds to the W
```

格式。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D°M'S.SSS"));
      st_aslatlon
-----
-2°19'29.928" -3°14'3.243"
```

格式。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
      st_aslatlon
-----
2.3250 degrees S 3.2342 degrees W
```

格式。

```
SELECT (ST_AsLatLonText('POINT (-302.2342342 -792.32498)'));
      st_aslatlon
-----
72°19'29.928"S 57°45'56.757"E
```

7.9.3.8 ST_AsMARC21

ST_AsMARC21 — Returns geometry as a MARC21/XML record with a geographic datafield (034).

Synopsis

text **ST_AsMARC21** (geometry geom , text format='hddmmss');



This function returns a MARC21/XML record with **Coded Cartographic Mathematical Data** representing the bounding box of a given geometry. The format parameter allows to encode the coordinates in subfields \$d,\$e,\$f and \$g in all formats supported by the MARC21/XML standard. Valid formats are:

- cardinal direction, degrees, minutes and seconds (default): hddmmss
- decimal degrees with cardinal direction: hddd.dddddd
- decimal degrees without cardinal direction: ddd.dddddd
- decimal minutes with cardinal direction: hddmm.mmmm
- decimal minutes without cardinal direction: dddmm.mmmm
- decimal seconds with cardinal direction: hddmmss.sss

The decimal sign may be also a comma, e.g. hddmm,mmm.

The precision of decimal formats can be limited by the number of characters after the decimal sign, e.g. hddmm.mm for decimal minutes with a precision of two decimals.

This function ignores the Z and M dimensions.

LOC MARC21/XML versions supported:

- **MARC21/XML 1.1**

Availability: 3.3.0



Note

This function does not support non lon/lat geometries, as they are not supported by the MARC21/XML standard (Coded Cartographic Mathematical Data).



Note

The MARC21/XML Standard does not provide any means to annotate the spatial reference system for Coded Cartographic Mathematical Data, which means that this information will be lost after conversion to MARC21/XML.



Converting a POINT to MARC21/XML formatted as hddmmss (default)

```
SELECT ST_AsMARC21('SRID=4326;POINT(-4.504289 54.253312)::geometry');

          st_asmarc21
-----
<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" ">
```

```

        <subfield code="a"
>a</subfield>
        <subfield code="d"
>W0043015</subfield>
        <subfield code="e"
>W0043015</subfield>
        <subfield code="f"
>N0541512</subfield>
        <subfield code="g"
>N0541512</subfield>
    </datafield>
</record>

```

Converting a POLYGON to MARC21/XML formatted in decimal degrees

```

SELECT ST_AsMARC21('SRID=4326;POLYGON((-4.5792388916015625  ↵
54.18172660239091,-4.56756591796875  ↵
54.196993557130355,-4.546623229980469  ↵
54.18313300502024,-4.5792388916015625 54.18172660239091))'::geometry,' ↵
hddd.dddd');

<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" ">
    <subfield code="a"
>a</subfield>
    <subfield code="d"
>W004.5792</subfield>
    <subfield code="e"
>W004.5466</subfield>
    <subfield code="f"
>N054.1970</subfield>
    <subfield code="g"
>N054.1817</subfield>
  </datafield>
</record>

```

Converting a GEOMETRYCOLLECTION to MARC21/XML formatted in decimal minutes. The geometries order in the MARC21/XML output correspond to their order in the collection.

```

SELECT ST_AsMARC21('SRID=4326;GEOMETRYCOLLECTION(POLYGON((13.1  ↵
52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1  ↵
52.38333333333333,13.1 52.65)),POINT(-4.5 54.25))'::geometry,'hddmmm. ↵
mmmm');

          st_asmarc21
-----
<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" ">
    <subfield code="a"
>a</subfield>
    <subfield code="d"
>E01307.0000</subfield>

```

```

        <subfield code="e"
>E01331.0000</subfield>
        <subfield code="f"
>N05240.0000</subfield>
        <subfield code="g"
>N05224.0000</subfield>
        </datafield>
        <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a"
>a</subfield>
        <subfield code="d"
>W00430.0000</subfield>
        <subfield code="e"
>W00430.0000</subfield>
        <subfield code="f"
>N05415.0000</subfield>
        <subfield code="g"
>N05415.0000</subfield>
        </datafield>
</record>

```

国国

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);

国国

Transforms a geometry into the coordinate space of a MVT (**Mapbox Vector Tile**) tile, clipping it to the tile bounds if required. The geometry must be in the coordinate system of the target map (using **ST_Transform** if needed). Commonly this is **Web Mercator** (SRID:3857).

The function attempts to preserve geometry validity, and corrects it if needed. This may cause the result geometry to collapse to a lower dimension.

The rectangular bounds of the tile in the target map coordinate space must be provided, so the geometry can be transformed, and clipped if required. The bounds can be generated using **ST_TileEnvelope**.

This function is used to convert geometry into the tile coordinate space required by **ST_AsMVT**.

geom is the geometry to transform, in the coordinate system of the target map.

bounds is the rectangular bounds of the tile in map coordinate space, with no buffer.

extent is the tile extent size in tile coordinate space as defined by the **MVT specification**. Defaults to 4096.

buffer is the buffer size in tile coordinate space for geometry clipping. Defaults to 256.

clip_geom is a boolean to control if geometries are clipped or encoded as-is. Defaults to true.

2.2.0 添加。



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.

例

```
SELECT ST_AsText(ST_AsMVTGeom(
  ST_GeomFromText('POLYGON ((0 0, 10 0, 10 5, 0 -5, 0 0))'),
  ST_MakeBox2D(ST_Point(0, 0), ST_Point(4096, 4096)),
  4096, 0, false));
          st_astext
-----
MULTIPOLYGON(((5 4096,10 4091,10 4096,5 4096)),((5 4096,0 4101,0 4096,5 4096)))
```

Canonical example for a Web Mercator tile using a computed tile bounds to query and clip geometry. This assumes the data.geom column has srid of 4326.

```
SELECT ST_AsMVTGeom(
  ST_Transform( geom, 3857 ),
  ST_TileEnvelope(12, 513, 412), extent =
> 4096, buffer =
> 64) AS geom
FROM data
WHERE geom && ST_Transform(ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096)),4326)
```

例

[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);
```

国国

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 国国国国国国国国国国国国.

国国

```
WITH mvtgeom AS
(
  SELECT ST_AsMVTGeom(geom, ST_TileEnvelope(12, 513, 412), extent =
> 4096, buffer =
> 64) AS geom, name, description
  FROM points_of_interest
  WHERE geom && ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096))
)
SELECT ST_AsMVT(mvtgeom.*)
FROM mvtgeom;
```



```
SELECT ST_AsSVG('MULTICURVE((5 5,3 5,3 3,0 3),
  CIRCULARSTRING(0 0,2 1,2 2))'::geometry, 0, 0);
st_assvg
-----
M 5 -5 L 3 -5 3 -3 0 -3 M 0 0 A 2 2 0 0 0 2 -2
```

Multi-surface

```
SELECT ST_AsSVG('MULTISURFACE(
  CURVEPOLYGON(CIRCULARSTRING(-2 0,-1 -1,0 0,1 -1,2 0,0 2,-2 0),
    (-1 0,0 0.5,1 0,0 1,-1 0)),
  ((7 8,10 10,6 14,4 11,7 8)))'::geometry, 0, 2);
st_assvg
-----
M -2 0 A 1 1 0 0 0 0 0 A 1 1 0 0 0 2 0 A 2 2 0 0 0 -2 0 Z
M -1 0 L 0 -0.5 1 0 0 -1 -1 0 Z
M 7 -8 L 10 -10 6 -14 4 -11 Z
```

7.9.3.12 ST_AsTWKB

ST_AsTWKB — 将 TWKB (Tiny Well-Known Binary) 格式化为文本。

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。它基于 Well-Known Binary (WKB) 格式，但更加紧凑。

该函数支持多种几何类型，包括点、线、面、多面体等。它可以根据需要返回带有大小和盒子的几何数据。精度参数 (prec, prec_z, prec_m) 用于指定返回结果的精度。

该函数还支持批量操作，可以将多个几何对象一次性转换为 TWKB 格式。批量操作时，需要提供几何对象的数组 (geom) 和它们的标识符 (ids)。

该函数还支持返回带有大小和盒子的几何数据。通过设置 with_sizes 和 with_boxes 参数，可以控制返回结果是否包含这些信息。此外，还可以通过 array_agg 函数将批量操作的结果聚合为一个数组。



Note

<https://github.com/TWKB/Specification> 提供了 TWKB 格式的完整规范。此外，还可以参考 <https://github.com/TWKB/twkb.js> 了解 TWKB 格式的实现细节。

Enhanced: 2.4.0 memory and speed improvements.

2.2.0 首次引入。

例

```
SELECT ST_AsTWKB('LINESTRING(1 1,5 5)::geometry');
          st_astwkb
-----
\x02000202020808
```

使用 `array_agg()` 聚合几何和标识符的 TWKB 二进制表示。

```
SELECT ST_AsTWKB(array_agg(geom), array_agg(gid)) FROM mytable;
          st_astwkb
-----
\x040402020400000202
```

例

`ST_GeomFromTWKB`, `ST_AsBinary`, `ST_AsEWKB`, `ST_AsEWKT`, `ST_GeomFromText`

7.9.3.13 ST_AsX3D

`ST_AsX3D` — 将 X3D XML 编码: ISO-IEC-19776-1.2-X3DEncodings-XML 转换为二进制。

Synopsis

text `ST_AsX3D`(geometry g1, integer maxdecimaldigits=15, integer options=0);

例

例 <http://www.web3d.org/standards/number/19776-1> 定义了 X3D XML 编码。maxdecimaldigits (默认为 15) 指定了最大小数位数。

Note



X3D 是 PostGIS 3.6.0 引入的。PostGIS 3.6.0 之前，X3D 是作为二进制数据存储在数据库中的。X3D 是 XML 格式，因此它需要被编码为二进制格式。PostGIS 3.6.0 引入了 `ST_AsX3D` 函数，用于将 X3D XML 编码为二进制格式。这允许 X3D 数据以文本格式存储在数据库中，并可以通过 `ST_AsX3D` 函数轻松转换为二进制格式。这解决了在 PostGIS 2D/3D 中存储 X3D 数据的问题 (bug ticket)。

' 0 ' 选项指定了 X/Y (或 X/Y/Z) 的格式。PostGIS 2.2 之前，X3D 使用 `long, lat` 或 `X, Y` 格式。PostGIS 3.6.0 引入了 `ST_AsX3D` 函数，用于将 X3D XML 编码为二进制格式。这允许 X3D 数据以文本格式存储在数据库中，并可以通过 `ST_AsX3D` 函数轻松转换为二进制格式。这解决了在 PostGIS 2D/3D 中存储 X3D 数据的问题 (bug ticket)。

- 0: 指定 X/Y (或 X/Y/Z) 的格式。PostGIS 2.2 之前，X3D 使用 `long, lat` 或 `X, Y` 格式。PostGIS 3.6.0 引入了 `ST_AsX3D` 函数，用于将 X3D XML 编码为二进制格式。这允许 X3D 数据以文本格式存储在数据库中，并可以通过 `ST_AsX3D` 函数轻松转换为二进制格式。这解决了在 PostGIS 2D/3D 中存储 X3D 数据的问题 (bug ticket)。
- 1: X 和 Y 的格式。指定 (GeoCoordinate) 格式。PostGIS 3.6.0 引入了 `ST_AsX3D` 函数，用于将 X3D XML 编码为二进制格式。这允许 X3D 数据以文本格式存储在数据库中，并可以通过 `ST_AsX3D` 函数轻松转换为二进制格式。这解决了在 PostGIS 2D/3D 中存储 X3D 数据的问题 (bug ticket)。
- 2: 指定 WGS84 (SRID 4326) 格式。PostGIS 3.6.0 引入了 `ST_AsX3D` 函数，用于将 X3D XML 编码为二进制格式。这允许 X3D 数据以文本格式存储在数据库中，并可以通过 `ST_AsX3D` 函数轻松转换为二进制格式。这解决了在 PostGIS 2D/3D 中存储 X3D 数据的问题 (bug ticket)。

PostGIS 几何类型	2D X3D 几何类型	3D X3D 几何类型
LINESTRING	LINESTRING 2D 几何类型。 PolyLine2D 几何类型。	LineSet
MULTILINESTRING	MULTILINESTRING 2D 几何类型。 PolyLine2D 几何类型。	IndexedLineSet
MULTIPOINT	PointSet	PointSet
POINT	POINT 2D 几何类型。	POINT 3D 几何类型。
(MULTI) POLYGON, POLYHEDRALSURFACE	(MULTI) POLYGON 3D 几何类型 (markup) 几何类型。	IndexedFaceSet (几何类型 几何类型 (faceset) 几何类型 几何类型。)
TIN	TriangleSet2D (几何类型 几何类型。)	IndexedTriangleSet



Note

2.     .     .     .

Lots of advancements happening in 3D space particularly with **X3D Integration with HTML5**

FreeWRL is a free, open source, cross platform, X3D viewer. It, along with, Free Wrl, is available at <http://freewrl.sourceforge.net/>. FreeWRL_Launcher is a Java application that runs FreeWRL.

Also check out [PostGIS minimalist X3D viewer](#) that utilizes this function and [x3dDom html/js open source toolkit](#).

2.0.0 0000 ISO-IEC-19776-1.2-X3DEncodings-XML 0000000000.

☐☐☐☐: 2.2.0 ☐☐☐☐☐☐☐☐☐☐ (x/y, ☐☐/☐☐) ☐☐☐☐☐☐☐☐☐☐. ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐.

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

00: 00000000 X3D 00000000. 00000 FreeWrl 000 X3D 0000000000000000
 00000000.

```
SELECT '<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d
-3.0.dtd">
<X3D>
  <Scene>
    <Transform>
      <Shape>
        <Appearance>
          <Material emissiveColor=''0 0 1''/>
        </Appearance>
      </Shape>
    </Transform>
  </Scene>
</X3D>
' ||
ST_AsX3D( ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ) ) ||
'</Shape>
</Transform>
' )
```

```

</Scene>
</X3D>
>' As x3ddoc;

    x3ddoc
    -----
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d ←
-3.0.dtd">
<X3D>
  <Scene>
    <Transform>
      <Shape>
        <Appearance>
          <Material emissiveColor='0 0 1' />
        </Appearance>
        <IndexedFaceSet coordIndex='0 1 2 3 -1 4 5 6 7 -1 8 9 10 11 -1 12 13 14 15 -1 16 17 ←
18 19 -1 20 21 22 23'>
          <Coordinate point='0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 ←
1 0 1 0 0 1 1 1 0 1 1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 ←
1 0 1 1' />
        </IndexedFaceSet>
      </Shape>
    </Transform>
  </Scene>
</X3D>

```

PostGIS buildings

Copy and paste the output of this query to [x3d scene viewer](#) and click Show

```

SELECT string_agg('<Shape
>' || ST_AsX3D(ST_Extrude(geom, 0,0, i*0.5)) ||
  '<Appearance>
    <Material diffuseColor="' || (0.01*i)::text || ' 0.8 0.2" specularColor="' || ←
    (0.05*i)::text || ' 0 0.5"/>
  </Appearance>
</Shape
>', '')
FROM ST_Subdivide(ST_Letters('PostGIS'),20) WITH ORDINALITY AS f(geom,i);

```



Buildings formed by subdividing PostGIS and extrusion

图 6-3: 通过 PostGIS 的 ST_Subdivide 函数和 ST_Extrude 函数生成 3D 建筑模型

```

SELECT ST_AsX3D(
  ST_Translate(
    ST_Force_3d(
      ST_Buffer(ST_Point(10,10),5, 'quad_segs=2')), 0,0,
      3)
    ,6) As x3dfrag;

x3dfrag
-----
<IndexedFaceSet coordIndex="0 1 2 3 4 5 6 7">
  <Coordinate point="15 10 3 13.535534 6.464466 3 10 5 3 6.464466 6.464466 3 5 10 3 6.464466 13.535534 3 10 15 3 13.535534 13.535534 3 " />
</IndexedFaceSet>

```

图 7.9.3.13: TIN

```

SELECT ST_AsX3D(ST_GeomFromEWKT('TIN (((
    0 0 0,
    0 0 1,
    0 1 0,
    0 0 0
  )), ((
    0 0 0,
    0 1 0,
    1 1 0,
    0 0 0
  ))
)')) As x3dfrag;

x3dfrag
-----
<IndexedTriangleSet index='0 1 2 3 4 5'
><Coordinate point='0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0' /></IndexedTriangleSet>

```

图 7.9.3.14: 折线 (折线集合)

```

SELECT ST_AsX3D(
  ST_GeomFromEWKT('MULTILINESTRING((20 0 10,16 -12 10,0 -16 10,-12 -12 10,-20 0 10,-12 16 10,0 24 10,16 16 10,20 0 10),
  (12 0 10,8 8 10,0 12 10,-8 8 10,-8 0 10,-8 -4 10,0 -8 10,8 -4 10,12 0 10)))')
) As x3dfrag;

x3dfrag
-----
<IndexedLineSet coordIndex='0 1 2 3 4 5 6 7 0 -1 8 9 10 11 12 13 14 15 8'>
  <Coordinate point='20 0 10 16 -12 10 0 -16 10 -12 -12 10 -20 0 10 -12 16 10 0 24 10 16 16 10 12 0 10 8 8 10 0 12 10 -8 8 10 -8 0 10 -8 -4 10 0 -8 10 8 -4 10 ' />
</IndexedLineSet>

```

7.9.3.14 ST_GeoHash

ST_GeoHash — 返回 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) );
```

```
  st_geohash
```

```
-----
c0w3hf1s70w3hf1s70w3
```

```
SELECT ST_GeoHash( ST_Point(-126,48), 5);
```

```
  st_geohash
```

```
-----
c0w3h
```

```
-- This line contains the point, so the GeoHash is a prefix of the point code
```

```
SELECT ST_GeoHash('LINESTRING(-126 48, -126.1 48.1)::geometry);
```

```
  st_geohash
```

```
-----
c0w3
```

国国

ST_GeomFromGeoHash, ST_PointFromGeoHash, ST_Box2dFromGeoHash

7.10 运算符 (operator)

7.10.1 Bounding Box Operators

7.10.1.1 &&

&& — A 2D 几何体 B 2D 几何体 TRUE 或 FALSE.

Synopsis

boolean **&&**(geometry A , geometry B);
boolean **&&**(geography A , geography B);

注意

&& 几何体 A 2D 几何体 B 2D 几何体 TRUE 或 FALSE.



Note

几何体 (operand) 必须是 2D 几何体.

注意: 2.0.0 版本 (polyhedral surface) 支持.

1.5.0 版本支持.

- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.

注意

```
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)

注意

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);

国国

The **&&** operator returns TRUE if the cached 2D bounding box of geometry A intersects the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

国国

```
SELECT ST_Point(1,1) && ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS overlaps;

overlaps
-----
t
(1 row)
```

国国

&&(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);

国国

The && operator returns TRUE if the 2D bounding box A intersects the cached 2D bounding box of geometry B, using float precision. This means that if A is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

**Note**

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

国国

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_Point(1,1) AS overlaps;
```

```
overlaps
-----
t
(1 row)
```

国国

**&&(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);

国国

The && operator returns TRUE if two 2D bounding boxes A and B intersect each other, using float precision. This means that if A (or B) is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

**Note**

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.

- ✓ This method supports Circular Strings and Curves.
- ✓ This function supports Polyhedral surfaces.

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

如果任一操作数 (operand) 是空几何体，则返回 TRUE。

2.0.0 版本引入。

- ✓ This method supports Circular Strings and Curves.
- ✓ This function supports Polyhedral surfaces.
- ✓ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✓ This function supports 3d and will not drop the z-index.

3D 重叠 (Z)

```
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 重叠 (ZM)

```
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

重叠

&&

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);

重叠

The **&&&** operator returns TRUE if the cached n-D bounding box of geometry A intersects the n-D bounding box B, using float precision. This means that if B is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)

**Note**

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



This function supports 3d and will not drop the z-index.

☒☒

```
SELECT ST_MakePoint(1,1,1) &&& ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) AS ↔
       overlaps;
```

```
overlaps
-----
t
(1 row)
```

☒☒

&&&(gidx,geometry), &&&(gidx,gidx)

7.10.1.7 &&&(gidx,geometry)

&&&(gidx,geometry) — Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.

Synopsis

boolean **&&&**(gidx A , geometry B);

☒☒

The **&&&** operator returns TRUE if the n-D bounding box A intersects the cached n-D bounding box of geometry B, using float precision. This means that if A is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)

**Note**

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



This function supports 3d and will not drop the z-index.

SQL

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_MakePoint(1,1,1) AS overlaps;
overlaps
-----
t
(1 row)
```

SQL

`&&&(geometry,gidx), &&&(gidx,gidx)`

7.10.1.8 **`&&&(gidx,gidx)`**

`&&&(gidx,gidx)` — Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

Synopsis

boolean **`&&&`**(gidx A , gidx B);

SQL

The **`&&&`** operator returns TRUE if two n-D bounding boxes A and B intersect each other, using float precision. This means that if A (or B) is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)



Note

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



This function supports 3d and will not drop the z-index.

例

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_3DMakeBox(ST_MakePoint(1,1,1), ST_MakePoint(3,3,3)) AS overlaps;

overlaps
-----
t
(1 row)
```

例

`&&&(geometry,gidx), &&&(gidx,geometry)`

7.10.1.9 &<

`&<` — A 与 B 相交是否返回 TRUE。

Synopsis

boolean **`&<`**(geometry A , geometry B);

例

`&<` 返回 A 与 B 是否相交，如果 B 与 A 相交，则返回 TRUE。



Note

如果 (operand) 与 B 相交，则返回 TRUE。

例

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &< tbl2.column2 AS overleft
FROM
  ( VALUES
    (1, 'LINESTRING(1 2, 4 6)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)::geometry),
    (3, 'LINESTRING(0 1, 0 5)::geometry),
    (4, 'LINESTRING(6 0, 6 1)::geometry)) AS tbl2;

column1 | column1 | overleft
-----+-----+-----
1 | 2 | f
1 | 3 | f
1 | 4 | t
(3 rows)
```

返回

&&, |&>, &>, &<|

7.10.1.10 &<|

&<| — A 与 B 是否相交。如果相交，返回 TRUE。

Synopsis

boolean **&<|**(geometry A , geometry B);

返回

&<| 检查 A 与 B 是否相交。如果相交，返回 TRUE。

✔ This method supports Circular Strings and Curves.

✔ This function supports Polyhedral surfaces.



Note

操作数 (operand) 必须是几何类型。

返回

```
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)

返回

&&, |&>, &>, &<|

7.10.1.11 &>

&> — A 是否包含 B。如果包含，返回 TRUE。

Synopsis

boolean **&>**(geometry A , geometry B);

描述

&> 当几何体 A 与几何体 B 相交时返回 TRUE，当几何体 B 完全包含在几何体 A 内部时返回 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 overright
FROM
  ( VALUES
    (1, 'LINESTRING(1 2, 4 6)::geometry') AS tbl1,
    ( VALUES
      (2, 'LINESTRING(0 0, 3 3)::geometry),
      (3, 'LINESTRING(0 1, 0 5)::geometry),
      (4, 'LINESTRING(6 0, 6 1)::geometry') AS tbl2;
```

column1	column1	overright
1	2	t
1	3	t
1	4	f

(3 rows)

描述

&&, **|&>**, **&<|**, **&<**

7.10.1.12 <<

<< — A 与 B 不相交时返回 TRUE。

Synopsis

boolean **<<**(geometry A , geometry B);

描述

<< 当几何体 A 与几何体 B 不相交时返回 TRUE。



Note

几何体 (operand) 必须是有效的几何体。

例

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 << tbl2.column2 AS left
FROM
  ( VALUES
    (1, 'LINESTRING (1 2, 1 5)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 3)::geometry),
    (3, 'LINESTRING (6 0, 6 5)::geometry),
    (4, 'LINESTRING (2 2, 5 6)::geometry)) AS tbl2;
```

column1	column1	left
1	2	f
1	3	t
1	4	t

(3 rows)

例

>>, |>>, <<|

7.10.1.13 <<|

<<| — A 与 B 的交集是否为空。TRUE 表示是。

Synopsis

boolean <<|(geometry A , geometry B);

例

<<| 与 A 的交集是否为空 B 的交集是否为空 TRUE 表示是。



Note

与 (operand) 的交集是否为空。

例

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 <<| tbl2.column2 AS below
FROM
  ( VALUES
    (1, 'LINESTRING (0 0, 4 3)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (1 4, 1 7)::geometry),
    (3, 'LINESTRING (6 1, 6 5)::geometry),
    (4, 'LINESTRING (2 3, 5 6)::geometry)) AS tbl2;
```

column1	column1	below
1	2	t

	1		3		f
	1		4		f
(3 rows)					

<<, >>, |>>

7.10.1.14 =

= — Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.

Synopsis

boolean =(geometry A , geometry B);
boolean =(geography A , geography B);

The = operator returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B. PostgreSQL uses the =, <, and > operators defined for geometries to perform internal orderings and comparison of geometries (ie. in a GROUP BY or ORDER BY clause).



Note
Only geometry/geography that are exactly equal in all respects, with the same coordinates, in the same order, are considered equal by this operator. For "spatial equality", that ignores things like coordinate order, and can detect features that cover the same spatial area with different representations, use [ST_OrderingEquals](#) or [ST_Equals](#)



Caution
This operand will NOT make use of any indexes that may be available on the geometries. For an index assisted exact equality test, combine = with &&.

Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use ~= instead.

- ✓ This method supports Circular Strings and Curves.
- ✓ This function supports Polyhedral surfaces.

```

SELECT 'LINESTRING(0 0, 0 1, 1 0)::geometry = 'LINESTRING(1 1, 0 0)::geometry';
?column?
-----
f
(1 row)

SELECT ST_AsText(column1)
FROM ( VALUES
      ('LINESTRING(0 0, 1 1)::geometry'),
      ('LINESTRING(1 1, 0 0)::geometry')) AS foo;
      st_astext
-----
LINESTRING(0 0,1 1)
LINESTRING(1 1,0 0)
(2 rows)

-- Note: the GROUP BY uses the "=" to compare for geometry equivalency.
SELECT ST_AsText(column1)
FROM ( VALUES
      ('LINESTRING(0 0, 1 1)::geometry'),
      ('LINESTRING(1 1, 0 0)::geometry')) AS foo
GROUP BY column1;
      st_astext
-----
LINESTRING(0 0,1 1)
LINESTRING(1 1,0 0)
(2 rows)

-- In versions prior to 2.0, this used to return true --
SELECT ST_GeomFromText('POINT(1707296.37 4820536.77)') =
      ST_GeomFromText('POINT(1707296.27 4820536.87)') As pt_intersect;

--pt_intersect --
f

```

ST Equals, ST OrderingEquals, $\sim =$

7.10.1.15 >>

>> - A ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ B ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ TRUE ☐ ☐ ☐ ☐ ☐.

Synopsis

```
boolean >>( geometry A , geometry B );
```


>> A B TRUE.



Note

Note

□□□□□ (operand) □□□□□□□□□□□□□□□□□□□□□□□□.

例

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2
>
> tbl2.column2 AS right
FROM
  ( VALUES
    (1, 'LINESTRING (2 3, 5 6)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (1 4, 1 7)::geometry),
    (3, 'LINESTRING (6 1, 6 5)::geometry),
    (4, 'LINESTRING (0 0, 4 3)::geometry)) AS tbl2;
```

column1	column1	right
1	2	t
1	3	f
1	4	f

(3 rows)

例

<<, |>>, <<|

7.10.1.16 @

@ — B 是否包含 A 是否包含 TRUE 或 FALSE。

Synopsis

boolean @(geometry A , geometry B);

例

@ 是否包含 B 是否包含 A 是否包含 TRUE 或 FALSE。



Note

几何体 (operand) 必须是有效的几何体。

例

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 @ tbl2.column2 AS contained
FROM
  ( VALUES
    (1, 'LINESTRING (1 1, 3 3)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 4)::geometry),
    (3, 'LINESTRING (2 2, 4 4)::geometry),
    (4, 'LINESTRING (1 1, 3 3)::geometry)) AS tbl2;
```

column1	column1	contained

-----+-----+-----			
	1	2	t
	1	3	f
	1	4	t
(3 rows)			

☒☒

~, &&

7.10.1.17 @ (geometry, box2df)

@ (geometry, box2df) — Returns TRUE if a geometry’s 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).

Synopsis

boolean @ (geometry A , box2df B);

☒☒

The @ operator returns TRUE if the A geometry’s 2D bounding box is contained the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

✔ This method supports Circular Strings and Curves.

✔ This function supports Polyhedral surfaces.

☒☒

SELECT ST_Buffer(ST_GeomFromText('POINT(2 2)'), 1) @ ST_MakeBox2D(ST_Point(0,0), ST_Point(↵ (5,5)) AS is_contained;	
is_contained	

t	
(1 row)	

☒☒

&&(geometry, box2df), &&(box2df, geometry), &&(box2df, box2df), ~(geometry, box2df), ~(box2df, geometry),
~(box2df, box2df), @(box2df, geometry), @(box2df, box2df)

7.10.1.18 @(box2df,geometry)

@(box2df,geometry) — Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.

Synopsis

boolean @(box2df A , geometry B);

注意

The @ operator returns TRUE if the 2D bounding box A is contained into the B geometry's 2D bounding box, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

注意

```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_Buffer(ST_GeomFromText('POINT(1 1)') ←
, 10) AS is_contained;
```

```
is_contained
-----
t
(1 row)
```

注意

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,box2df)

7.10.1.19 @(box2df,box2df)

@(box2df,box2df) — Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.

Synopsis

boolean @(box2df A , box2df B);

The @ operator returns TRUE if the 2D bounding box A is contained into the 2D bounding box B, using float precision. This means that if A (or B) is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexas (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) AS is contained;
```

```
is_contained
-----
t
(1 row)
```



$\&(\text{geometry}, \text{box2df}), \&\&(\text{box2df}, \text{geometry}), \&\&(\text{box2df}, \text{box2df}), \sim(\text{geometry}, \text{box2df}), \sim(\text{box2df}, \text{geometry}),$
 $\sim(\text{box2df}, \text{box2df}), @(\text{geometry}, \text{box2df}), @(\text{box2df}, \text{geometry})$

7.10.1.20 |&>

|&> - A ☐ ☐ ☐ ☐ ☐ ☐ B ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ TRUE ☐ ☐ ☐ ☐ ☐.

Synopsis

```
boolean |&>( geometry A , geometry B );
```



```
&> A <- B, TRUE
```



Note

(operand)

例

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |&
> tbl2.column2 AS overabove
FROM
  ( VALUES
    (1, 'LINESTRING(6 0, 6 4)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)::geometry),
    (3, 'LINESTRING(0 1, 0 5)::geometry),
    (4, 'LINESTRING(1 2, 4 6)::geometry)) AS tbl2;
```

column1	column1	overabove
1	2	t
1	3	f
1	4	f

(3 rows)

例

&&, &>, &<|, &<

7.10.1.21 |>>

|>> — A 是否完全在 B 上方。如果为 TRUE，则返回 TRUE。

Synopsis

boolean |>>(geometry A , geometry B);

例

The |>> operator returns TRUE if the bounding box of geometry A is strictly above the bounding box of geometry B.



Note

如果 A 的 bounding box 完全在 B 的 bounding box 上方，则返回 TRUE。

例

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |>> tbl2.column2 AS above
FROM
  ( VALUES
    (1, 'LINESTRING (1 4, 1 7)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 2)::geometry),
    (3, 'LINESTRING (6 1, 6 5)::geometry),
    (4, 'LINESTRING (2 3, 5 6)::geometry)) AS tbl2;
```

column1	column1	above

-----+-----+-----		
	1	2 t
	1	3 f
	1	4 f
(3 rows)		

函数

<<, >>, <<|

7.10.1.22 ~

~ — A 与 B 不相交是否返回 TRUE 值。

Synopsis

boolean ~(geometry A , geometry B);

函数

~ 返回 A 与 B 不相交是否返回 TRUE 值。



Note

operator (operand) 返回不相交是否返回 TRUE 值。

函数

```
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)		

函数

@, &&

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);

国国

The ~ operator returns TRUE if the 2D bounding box of a geometry A contains the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

**Note**

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

国国

```
SELECT ST_Buffer(ST_GeomFromText('POINT(1 1)'), 10) ~ ST_MakeBox2D(ST_Point(0,0), ST_Point(↵
(2,2)) AS contains;
```

```
contains
-----
t
(1 row)
```

国国

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(box2df,geometry), ~(box2df,box2df),
 @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.24 ~(box2df,geometry)

~(box2df,geometry) — Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.

Synopsis

boolean ~(box2df A , geometry B);



The `~` operator returns TRUE if the 2D bounding box A contains the B geometry's bounding box, using float precision. This means that if A is a (double precision) `box2d`, it will be internally converted to a float precision 2D bounding box (`BOX2DF`)

**Note**

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_Buffer(ST_GeomFromText('POINT(2 2)') ←
, 1) AS contains;

contains
-----
t
(1 row)
```



`&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,box2df),
@(geometry,box2df), @(box2df,geometry), @(box2df,box2df)`

7.10.1.25 ~(box2df,box2df)

`~(box2df,box2df)` — Returns TRUE if a 2D float precision bounding box (`BOX2DF`) contains another 2D float precision bounding box (`BOX2DF`).

Synopsis

```
boolean ~( box2df A , box2df B );
```



The `~` operator returns TRUE if the 2D bounding box A contains the 2D bounding box B, using float precision. This means that if A is a (double precision) `box2d`, it will be internally converted to a float precision 2D bounding box (`BOX2DF`)

**Note**

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

例

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) AS contains;
```

```
contains
-----
t
(1 row)
```

例

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [@&&\(geometry,box2df\)](#), [@&&\(box2df,geometry\)](#), [@&&\(box2df,box2df\)](#)

7.10.1.26 ~=

`~=` — A 与 B 是否相等 TRUE 或 FALSE。

Synopsis

boolean `~=`(geometry A , geometry B);

例

`~=` 返回 TRUE/FALSE A 与 B 是否相等 TRUE 或 FALSE。



Note

运算符 (operand) 必须为几何类型。

1.5.0 版本行为更改。



This function supports Polyhedral surfaces.



Warning

This operator has changed behavior in PostGIS 1.5 from testing for actual geometric equality to only checking for bounding box equality. To complicate things it also depends on if you have done a hard or soft upgrade which behavior your database has. To find out which behavior your database has you can run the query below. To check for true equality use [ST_OrderingEquals](#) or [ST_Equals](#).

SQL

```
select 'LINESTRING(0 0, 1 1)::geometry' ~= 'LINESTRING(0 1, 1 0)::geometry' as equality;
equality      |
-----+-----
t             |
```

SQL

ST_Equals, ST_OrderingEquals, =

7.10.2 运算符 (operator)

7.10.2.1 <->

<-> — A 与 B 的最近邻接距离。

Synopsis

```
double precision <->( geometry A , geometry B );
double precision <->( geography A , geography B );
```

SQL

<-> 返回最近邻接距离。 "ORDER BY" 子句可用于索引辅助 (index-assisted) 最近邻接 (nearest neighbor) 查询。 PostgreSQL 9.5 引入了距离球 (distance sphere) 查询，9.5 引入了 KNN (K nearest neighbor) 查询。



Note

操作数 (operand) 必须是 2 个 GiST 索引。 ORDER BY 子句可用于索引辅助 (index-assisted) 最近邻接 (nearest neighbor) 查询。



Note

在 PostgreSQL 9.5 中，使用 a.geom 'SRID=3005;POINT(1011102 450541)::geometry' 查询，(子查询/CTE(common table expression) 子句) 可用于索引辅助 (index-assisted) 最近邻接 (nearest neighbor) 查询。

OpenGeo workshop: Nearest-Neighbour Searching 文档。

2.2.0 版本 -- PostgreSQL 9.5 引入了 KNN ("K nearest neighbor") 查询。 KNN 查询可用于索引辅助 (index-assisted) 最近邻接 (nearest neighbor) 查询。 PostgreSQL 9.4 引入了距离球 (distance sphere) 查询。

2.2.0 版本 -- PostgreSQL 9.5 引入了混合语法 (Hybrid syntax) 查询。 PostgreSQL 2.2 版本， PostgreSQL 9.5 引入了混合语法 (Hybrid syntax) 查询。

2.0.0 版本 -- KNN 查询可用于索引辅助 (index-assisted) 最近邻接 (nearest neighbor) 查询。 PostgreSQL 9.1 版本引入了距离球 (distance sphere) 查询。


```
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

KNN

```
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)		

XXXXXXXXXXXXXXXX"EXPLAIN ANALYZE" XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.

PostgreSQL 9.5 数据库, 数据库, 数据库. 数据库 KNN 数据库 CTE(common table expression) 数据库, 数据库 数据库:

```
WITH index_query AS (
  SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)>::geometry) as d,edabbr, vaabbr
    FROM va2005
 ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)>::geometry LIMIT 100)
SELECT *
  FROM index_query
 ORDER BY d limit 10;
```

d	edabbr	vaabbr
0	ALQ	128
5541.57712511724	ALQ	129A
5579.67450712005	ALQ	001
6083.4207708641	ALQ	131
7691.2205404848	ALQ	003

```

7900.75451037313 | ALQ | 122
8694.20710669982 | ALQ | 129B
9564.24289057111 | ALQ | 130
12089.665931705 | ALQ | 127
18472.5531479404 | ALQ | 002
(10 rows)

```

函数

ST_DWithin, ST_Distance, <#>

7.10.2.2 |=|

|=| — A 与 B 的最接近点 (closest point of approach) 的轨迹 (trajectory)。

Synopsis

double precision **|=|**(geometry A , geometry B);

函数

|=| 函数 (**ST_IsValidTrajectory** 函数) 返回 3 个值。返回 **ST_DistanceCPA** 函数值, 返回 **ST_DistanceCPA** 函数值 (PostgreSQL 9.5.0 版本) N 个最近邻居 (nearest neighbor) 的轨迹。



Note

函数 (operand) 返回 N 个 GiST 索引。返回 **ORDER BY** 函数值。



Note

函数, 函数 a.geom 函数 'SRID=3005;LINESTRINGM(0 0 0,0 0 1)::geometry' 函数, (函数/CTE(common table expression) 函数) 函数。

2.2.0 版本。 PostgreSQL 9.5 版本 (index-supported) 函数。

函数

```

-- Save a literal query trajectory in a psql variable...
\set qt 'ST_AddMeasure(ST_MakeLine(ST_MakePointM(-350,300,0),ST_MakePointM(-410,490,0)) ←
,10,20)'
-- Run the query !
SELECT track_id, dist FROM (
  SELECT track_id, ST_DistanceCPA(tr,:qt) dist
  FROM trajectories
  ORDER BY tr |=| :qt

```

```

LIMIT 5
) foo;
track_id      dist
-----+-----
395 | 0.576496831518066
380 | 5.06797130410151
390 | 7.72262293958322
385 | 9.8004461358071
405 | 10.9534397988433
(5 rows)

```


ST DistanceCPA, ST ClosestPointOfApproach, ST IsValidTrajectory

7.10.2.3 <#>

$$\langle \# \rangle = A \otimes B \otimes \otimes \otimes \otimes \otimes \otimes \otimes 2 \otimes \otimes \otimes \otimes \otimes \otimes \otimes \otimes \otimes.$$

Synopsis

```
double precision <#>( geometry A , geometry B );
```



<#> floating point . (PostgreSQL 9.1)
) . .



Note!

Note

ORDER BY `expression`.



Note

```

g1.geom <#> ORDER BY (ST_GeomFromText('POINT(1 2)')
<#> geom)

```

2.0.0. PostgreSQL 9.1 KNN.



```
SELECT *
FROM (
SELECT b.tlid, b.mtfcc,
       b.geom <#
> ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
                             745787 2948499,745740 2948468,745712 2948438,
                             745690 2948384,745677 2948319)',2249) As b dist,
```



```

        ST_Distance(b.geom, ST_GeomFromText('LINESTRING(746149 2948672,745954
        2948576,
        745787 2948499,745740 2948468,745712 2948438,
        745690 2948384,745677 2948319)',2249)) As act_dist
FROM bos_roads As b
ORDER BY b_dist, b.tlid
LIMIT 100) As foo
ORDER BY act_dist, tlid LIMIT 10;

```

tlid	mtfcc	b_dist	act_dist
85732027	S1400	0	0
85732029	S1400	0	0
85732031	S1400	0	0
85734335	S1400	0	0
85736037	S1400	0	0
624683742	S1400	0	128.528874268666
85719343	S1400	260.839270432962	260.839270432962
85741826	S1400	164.759294123275	260.839270432962
85732032	S1400	277.75	311.830282365264
85735592	S1400	222.25	311.830282365264



ST DWithin, ST Distance, <->

7.10.2.4 <<->>

`<->` — Returns the n-D distance between the A and B geometries or bounding boxes

Synopsis

```
double precision <<->>( geometry A , geometry B );
```



<<-> □□□□□□□□□□□□□□ N □ (□□□□) □□□□□□□□. □□□ □□□□□□□□
□□□□□□□□□□.

Note!

Note

```
[REDACTED] (operand) [REDACTED] N [REDACTED] GiST [REDACTED].
```


Note!

Note

1. `SELECT a.geom FROM a.geom WHERE SRID=3005;POINT(1011102 450541)::geometry`
 2. `(SELECT/CTE(common table expression) )`
 3.

2.2.0 2017-09-08. PostgreSQL 9.1 2012-06-26 KNN 2012-06-26.

☒☒

<->

7.11 Spatial Relationships

7.11.1 Topological Relationships

7.11.1.1 ST_3DIntersects

ST_3DIntersects — Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)

Synopsis

boolean **ST_3DIntersects**(geometry geomA , geometry geomB);

☒☒

Overlaps, Touches, Within all imply spatial intersection. If any of the aforementioned returns true, then the geometries also spatially intersect. Disjoint implies false for spatial intersection.



Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.



Note

Because of floating robustness failures, geometries don't always intersect as you'd expect them to after geometric processing. For example the closest point on a linestring to a geometry may not lie on the linestring. For these kind of issues where a distance of a centimeter you want to just consider as intersecting, use [ST_3DDWithin](#).

Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs.

2.0.0 ☒☒☒☒☒☒☒☒☒☒.



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

SQL

```
SELECT ST_3DIntersects(pt, line), ST_Intersects(pt, line)
FROM (SELECT 'POINT(0 0 2)::geometry As pt, 'LINESTRING (0 0 1, 0 2 3)::geometry As line) As foo;
st_3dintersects | st_intersects
-----+-----
f                | t
(1 row)
```

TIN Examples

```
SELECT ST_3DIntersects('TIN(((0 0 0,1 0 0,0 1 0,0 0 0)))::geometry, 'POINT(.1 .1 0)::geometry);
st_3dintersects
-----
t
```

SQL

ST_3DDWithin, ST_Intersects

7.11.1.2 ST_Contains

ST_Contains — Tests if every point of B lies in A, and their interiors have a point in common

Synopsis

boolean **ST_Contains**(geometry geomA, geometry geomB);

SQL

Returns TRUE if geometry A contains geometry B. A contains B if and only if all points of B lie inside (i.e. in the interior or boundary of) A (or equivalently, no points of B lie in the exterior of A), and the interiors of A and B have at least one point in common.

In mathematical terms: $ST\_Contains(A, B) \Leftrightarrow (A \sqcap B = B) \wedge (Int(A) \sqcap Int(B) \neq \emptyset)$

The contains relationship is reflexive: every geometry contains itself. (In contrast, in the **ST_ContainsProperly** predicate a geometry does *not* properly contain itself.) The relationship is antisymmetric: if **ST_Contains**(A, B) = true and **ST_Contains**(B, A) = true, then the two geometries must be topologically equal (**ST_Equals**(A, B) = true).

ST_Contains is the converse of **ST_Within**. So, **ST_Contains**(A, B) = **ST_Within**(B, A).



Note

Because the interiors must have a common point, a subtlety of the definition is that polygons and lines do *not* contain lines and points lying fully in their boundary. For further details see **Subtleties of OGC Covers, Contains, Within**. The **ST_Covers** predicate provides a more inclusive relationship.

**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Contains`.

GEOS 简体中文

Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.

**Important**

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

**Important**

Do not use this function with invalid geometries. You will get unexpected results.

NOTE: this is the "allowable" version that returns a boolean, not an integer.



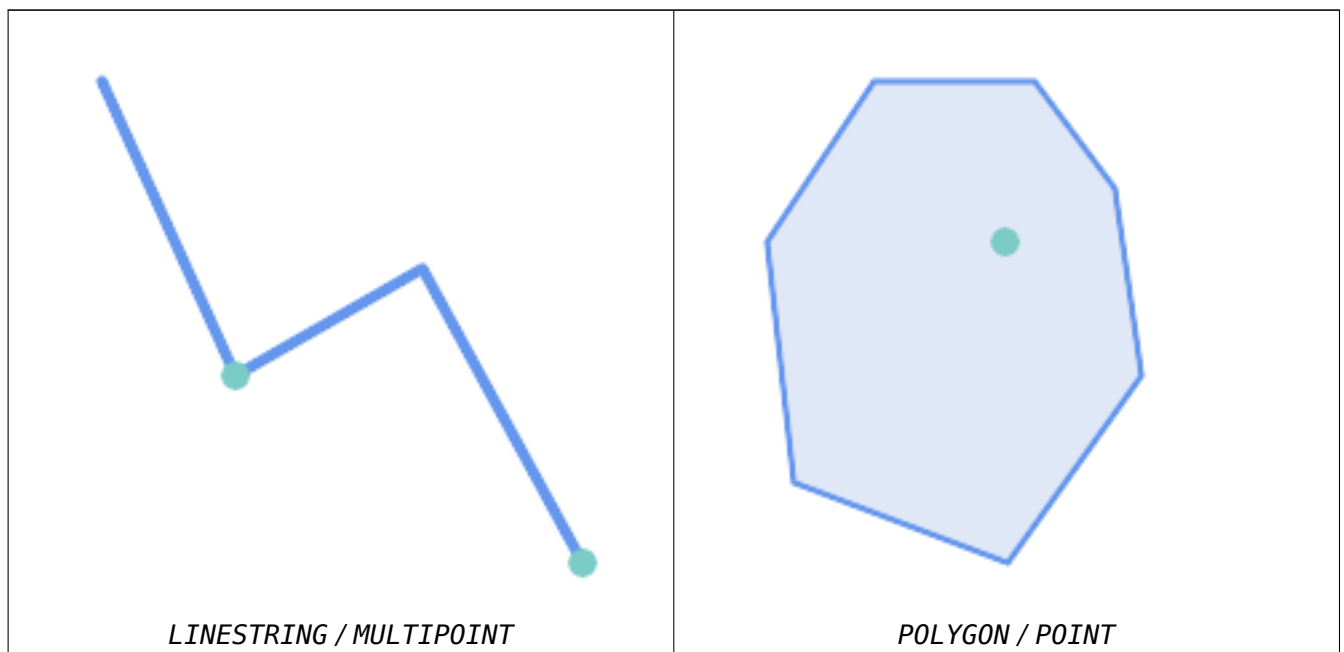
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3](#) - same as `within(geometry B, geometry A)`

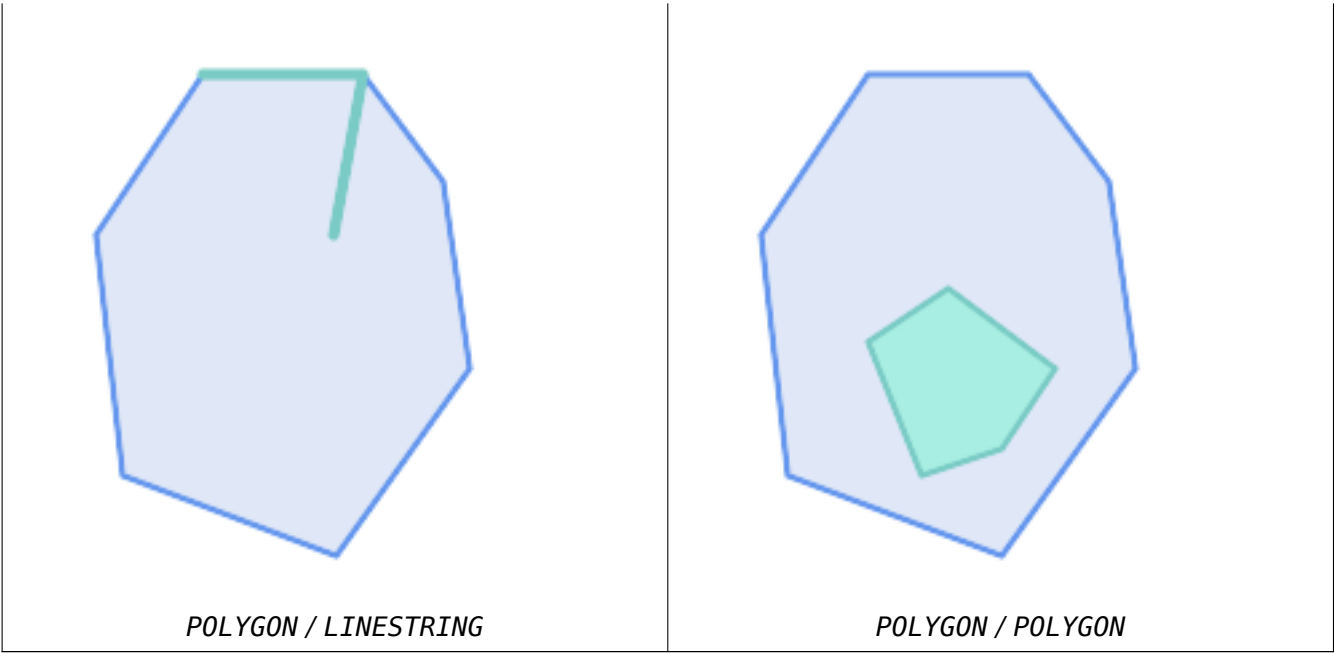


This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

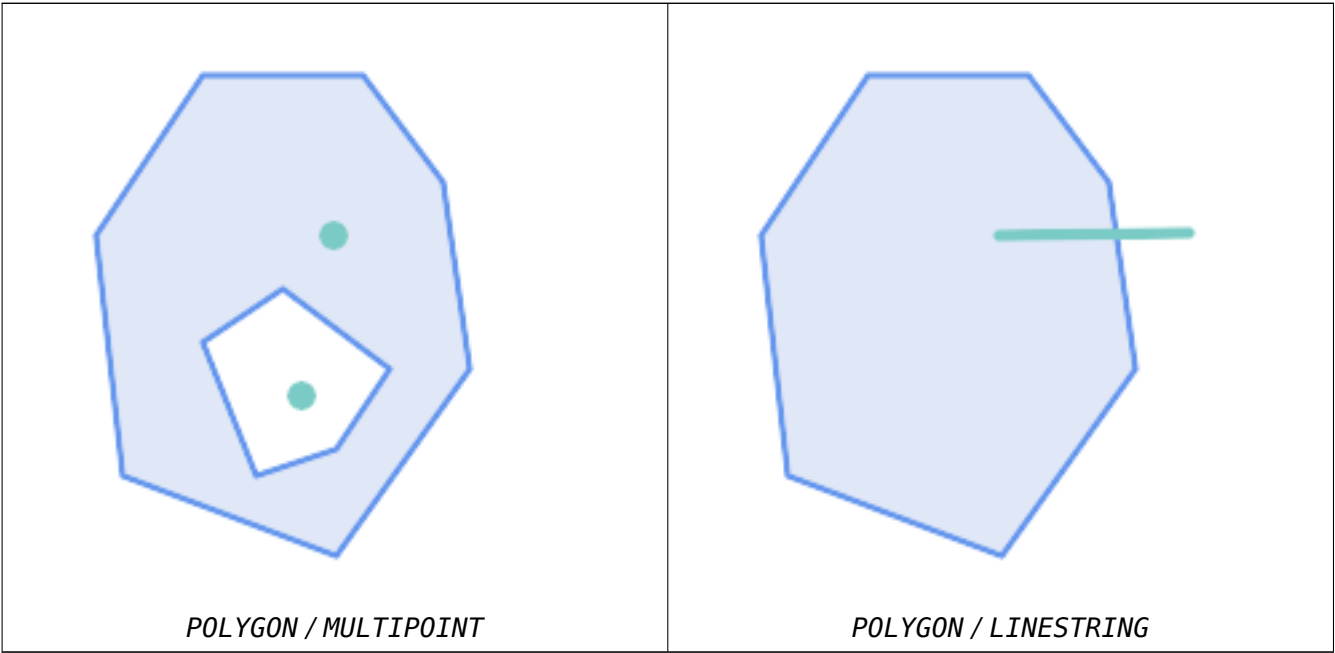
图例

`ST_Contains` returns TRUE in the following situations:

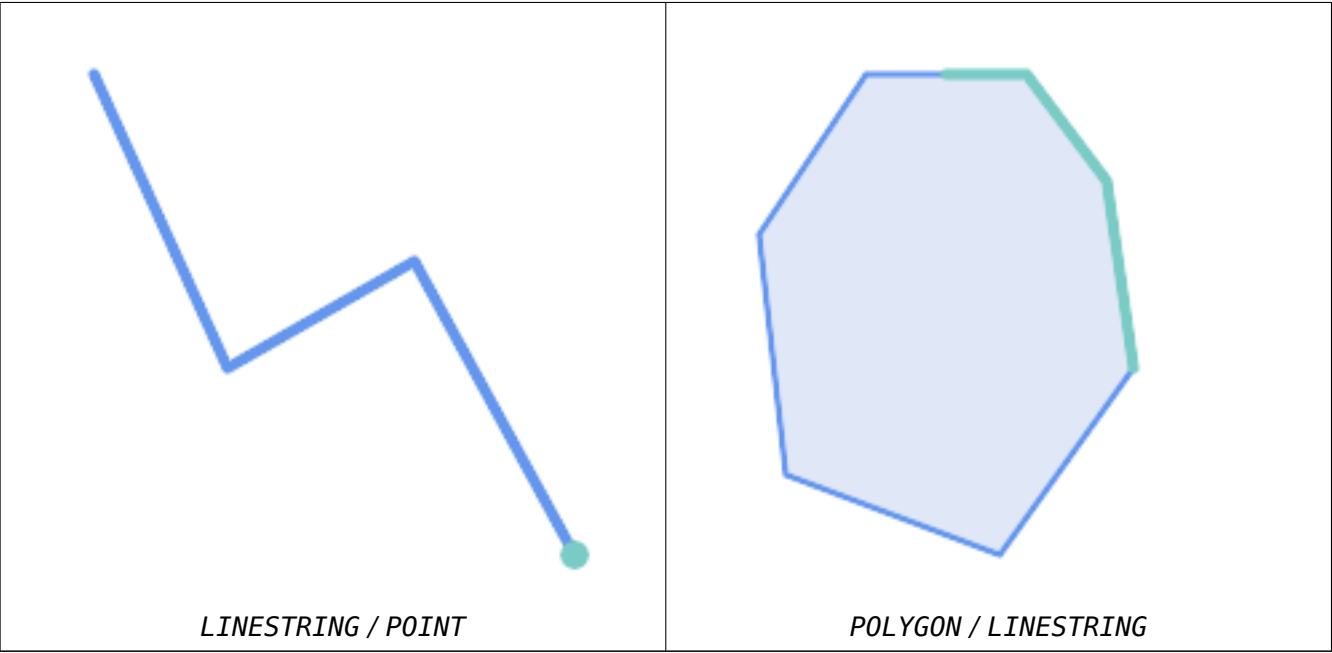




ST_Contains returns FALSE in the following situations:



Due to the interior intersection condition ST_Contains returns FALSE in the following situations (whereas ST_Covers returns TRUE):



```
-- A circle within a circle
SELECT ST_Contains(smallc, bigc) As smallcontainsbig,
       ST_Contains(bigc,smallc) As bigcontainssmall,
       ST_Contains(bigc, ST_Union(smallc, bigc)) as bigcontainsunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
         ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;

-- Result
smallcontainsbig | bigcontainssmall | bigcontainsunion | bigisunion | bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----+-----+-----
f                | t                | t                | t          | t                | f

-- Example demonstrating difference between contains and contains properly
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa, ST_ContainsProperly(geomA, geomA) AS acontainspropa,
       ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA, ST_Boundary(geomA)) As acontainspropba
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
            ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
            ( ST_Point(1,1) )
      ) As foo(geomA);

geomtype | acontainsa | acontainspropa | acontainsba | acontainspropba
-----+-----+-----+-----+-----
ST_Polygon | t         | f             | f           | f
ST_LineString | t        | f             | f           | f
ST_Point | t         | t             | f           | f
```

☒☒

[ST_Boundary](#), [ST_ContainsProperly](#), [ST_Covers](#), [ST_CoveredBy](#), [ST_Equals](#), [ST_Within](#)

7.11.1.3 ST_ContainsProperly

`ST_ContainsProperly` — Tests if every point of B lies in the interior of A

Synopsis

boolean **ST_ContainsProperly**(geometry geomA, geometry geomB);

☒☒

Returns true if every point of B lies in the interior of A (or equivalently, no point of B lies in the the boundary or exterior of A).

In mathematical terms: $ST\_ContainsProperly(A, B) \Leftrightarrow Int(A) \supset B = B$

A contains B properly if the DE-9IM Intersection Matrix for the two geometries matches [T**FF*FF*]

A does not properly contain itself, but does contain itself.

A use for this predicate is computing the intersections of a set of geometries with a large polygonal geometry. Since intersection is a fairly slow operation, it can be more efficient to use `containsProperly` to filter out test geometries which lie fully inside the area. In these cases the intersection is known a priori to be exactly the original test geometry.



Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_ContainsProperly`.



Note

The advantage of this predicate over [ST_Contains](#) and [ST_Intersects](#) is that it can be computed more efficiently, with no need to compute topology at individual points.

GEOS 3.10.0

1.4.0 简体中文.



Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



Important

Do not use this function with invalid geometries. You will get unexpected results.

☒☒

```
--a circle within a circle
SELECT ST_ContainsProperly(smallc, bigc) As smallcontainspropbig,
       ST_ContainsProperly(bigc,smallc) As bigcontainspropsmall,
       ST_ContainsProperly(bigc, ST_Union(smallc, bigc)) as bigcontainspropunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_ContainsProperly(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallcontainspropbig | bigcontainspropsmall | bigcontainspropunion | bigisunion | ↵
bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----+-----+-----+
f                    | t                    | f                    | t          | ↵
                    | f                    |                      |            |

--example demonstrating difference between contains and contains properly
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa, ↵
       ST_ContainsProperly(geomA, geomA) AS acontainspropa,
       ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA, ↵
       ST_Boundary(geomA)) As acontainspropba
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
            ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
            ( ST_Point(1,1) )
      ) As foo(geomA);

geomtype | acontainsa | acontainspropa | acontainsba | acontainspropba
-----+-----+-----+-----+-----+
ST_Polygon | t         | f             | f           | f
ST_LineString | t        | f             | f           | f
ST_Point | t         | t             | f           | f
```

☒☒

[ST_GeometryType](#), [ST_Boundary](#), [ST_Contains](#), [ST_Covers](#), [ST_CoveredBy](#), [ST_Equals](#), [ST_Relate](#), [ST_Within](#)

7.11.1.4 ST_CoveredBy

ST_CoveredBy — Tests if every point of A lies in B

Synopsis

boolean **ST_CoveredBy**(geometry geomA, geometry geomB);
boolean **ST_CoveredBy**(geography geogA, geography geogB);

☒☒

Returns true if every point in Geometry/Geography A lies inside (i.e. intersects the interior or boundary of) Geometry/Geography B. Equivalently, tests that no point of A lies outside (in the exterior of) B.

In mathematical terms: $ST\_CoveredBy(A, B) \Leftrightarrow A \sqcap B = A$

`ST_CoveredBy` is the converse of `ST_Covers`. So, `ST_CoveredBy(A,B) = ST_Covers(B,A)`.

Generally this function should be used instead of `ST_Within`, since it has a simpler definition which does not have the quirk that "boundaries are not within their geometry".



Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_CoveredBy`.



Important

Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`



Important

Do not use this function with invalid geometries. You will get unexpected results.

GEOS 简体中文

1.2.2 简体中文.

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Not an OGC standard, but Oracle has it too.

SQL

```
--a circle coveredby a circle
SELECT ST_CoveredBy(smallc,smallc) As smallinsmall,
       ST_CoveredBy(smallc, bigc) As smallcoveredbybig,
       ST_CoveredBy(ST_ExteriorRing(bigc), bigc) As exteriorcoveredbybig,
       ST_Within(ST_ExteriorRing(bigc),bigc) As exeriorwithinbig
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallinsmall | smallcoveredbybig | exteriorcoveredbybig | exeriorwithinbig
-----+-----+-----+-----
t            | t                  | t                    | f
(1 row)
```

SQL

`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);
```

描述

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 函数参考

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 函数参考.

1.2.2 函数参考.

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Not an OGC standard, but Oracle has it too.

示例

Geometry example

```
--a circle covering a circle
SELECT ST_Covers(smallc,smallc) As smallinsmall,
       ST_Covers(smallc, bigc) As smallcoversbig,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
```

```
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
        ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallinsmall | smallcoversbig | bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----
t             | f               | t                 | f
(1 row)
```

Geography Example

```
-- a point with a 300 meter buffer compared to a point, a point and its 10 meter buffer
SELECT ST_Covers(geog_poly, geog_pt) As poly_covers_pt,
        ST_Covers(ST_Buffer(geog_pt,10), geog_pt) As buff_10m_covers_cent
FROM (SELECT ST_Buffer(ST_GeogFromText('SRID=4326;POINT(-99.327 31.4821)'), 300) As
        geog_poly,
        ST_GeogFromText('SRID=4326;POINT(-99.33 31.483)') As geog_pt ) As foo;

poly_covers_pt | buff_10m_covers_cent
-----+-----
f               | t
```

ST_Contains, ST_CoveredBy, ST_Within

7.11.1.6 ST_Crosses

ST_Crosses — Tests if two geometries have some, but not all, interior points in common

Synopsis

boolean **ST_Crosses**(geometry g1, geometry g2);

Compares two geometry objects and returns true if their intersection "spatially crosses"; that is, the geometries have some, but not all interior points in common. The intersection of the interiors of the geometries must be non-empty and must have dimension less than the maximum dimension of the two input geometries, and the intersection of the two geometries must not equal either geometry. Otherwise, it returns false. The crosses relation is symmetric and irreflexive.

In mathematical terms: $ST\_Crosses(A, B) \Leftrightarrow (dim(Int(A) \cap Int(B)) < \max(dim(Int(A)), dim(Int(B))) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$

Geometries cross if their DE-9IM Intersection Matrix matches:

- T*T***** for Point/Line, Point/Area, and Line/Area situations
- T*****T** for Line/Point, Area/Point, and Area/Line situations
- 0***** for Line/Line situations
- the result is false for Point/Point and Area/Area situations

**Note**

The OpenGIS Simple Features Specification defines this predicate only for Point/Line, Point/Area, Line/Line, and Line/Area situations. JTS / GEOS extends the definition to apply to Line/Point, Area/Point and Area/Line situations as well. This makes the relation symmetric.

**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

**Important**

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



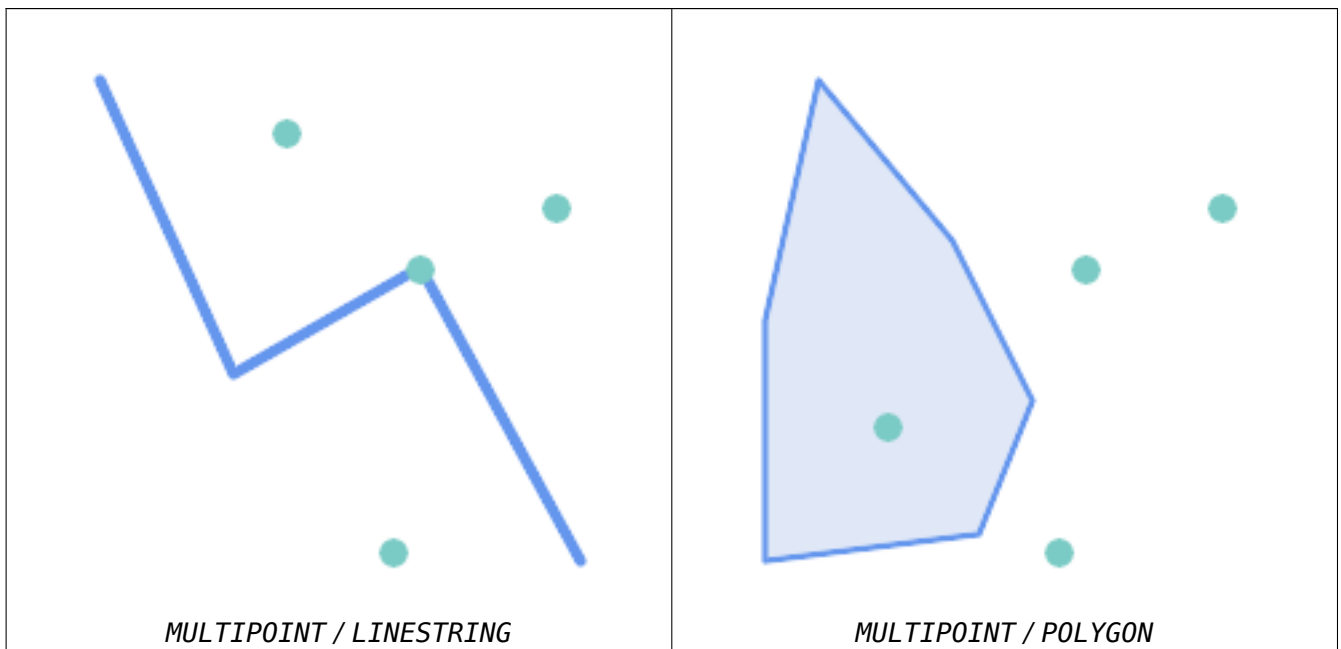
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.13.3](#)

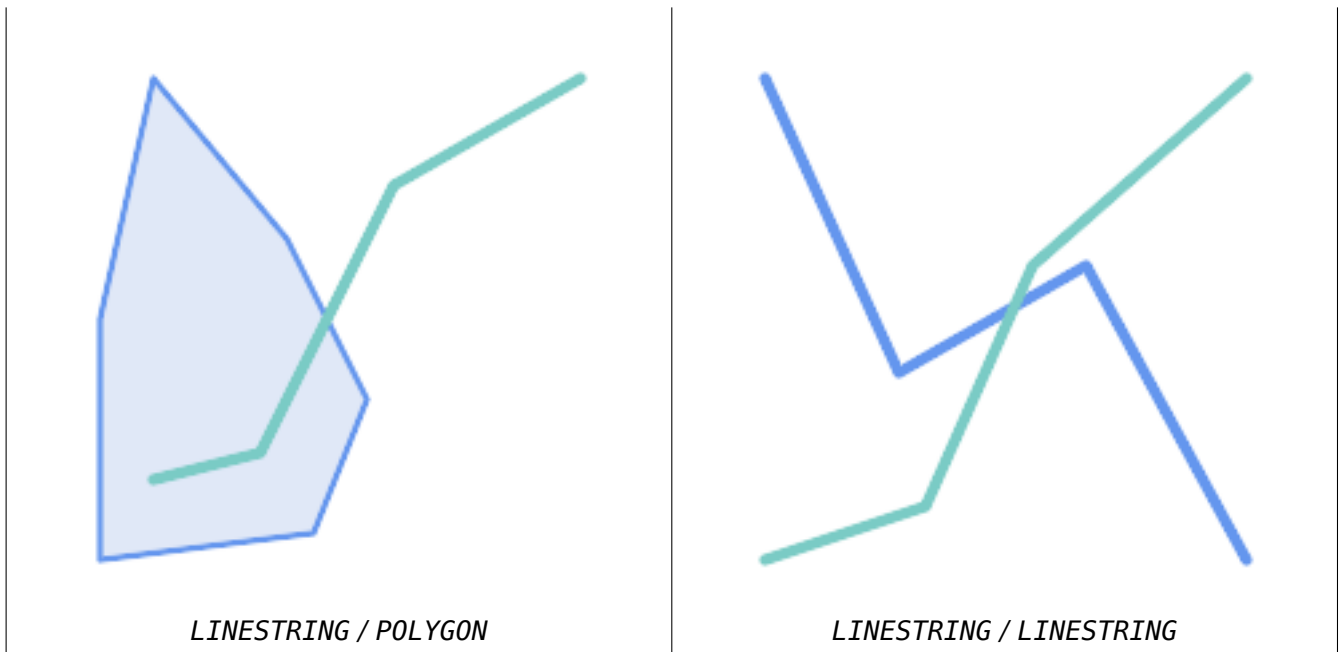


This method implements the SQL/MM specification. SQL-MM 3: 5.1.29



The following situations all return `true`.





Consider a situation where a user has two tables: a table of roads and a table of highways.

```
CREATE TABLE roads (
  id serial NOT NULL,
  geom geometry,
  CONSTRAINT roads_pkey PRIMARY KEY ( ↵
    road_id)
);
```

```
CREATE TABLE highways (
  id serial NOT NULL,
  the_geom geometry,
  CONSTRAINT roads_pkey PRIMARY KEY ( ↵
    road_id)
);
```

To determine a list of roads that cross a highway, use a query similar to:

```
SELECT roads.id
FROM roads, highways
WHERE ST_Crosses(roads.geom, highways.geom);
```

☒☒

ST_Contains, ST_Overlaps

7.11.1.7 ST_Disjoint

ST_Disjoint — Tests if two geometries have no points in common

Synopsis

boolean **ST_Disjoint**(geometry A , geometry B);

関数

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 関数



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

関数

```
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
st_disjoint
-----
t
(1 row)
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
st_disjoint
-----
f
(1 row)
```

関数

`ST_Intersects`

7.11.1.8 ST_Equals

`ST_Equals` — Tests if two geometries include the same set of points

Synopsis

boolean **ST_Equals**(geometry A, geometry B);

ⓘ

Returns true if the given geometries are “topologically equal”. Use this for a ‘better’ answer than ‘=’. Topological equality means that the geometries have the same dimension, and their point-sets occupy the same space. This means that the order of vertices may be different in topologically equal geometries. To verify the order of points is consistent use **ST_OrderingEquals** (it must be noted ST_OrderingEquals is a little more stringent than simply verifying order of points are the same).

In mathematical terms: $ST\_Equals(A, B) \Leftrightarrow A = B$

The following relation holds: $ST\_Equals(A, B) \Leftrightarrow ST\_Within(A,B) \wedge ST\_Within(B,A)$



Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



This method implements the **OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2**



This method implements the SQL/MM specification. SQL-MM 3: 5.1.24

Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal

ⓘ

```
SELECT ST_Equals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_equals
-----
t
(1 row)

SELECT ST_Equals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_equals
-----
t
(1 row)
```

ⓘ

ST_IsValid, ST_OrderingEquals, ST_Reverse, ST_Within

7.11.1.9 ST_Intersects

ST_Intersects — Tests if two geometries intersect (they have at least one point in common)

Synopsis

```
boolean ST_Intersects( geometry geomA , geometry geomB );
boolean ST_Intersects( geography geogA , geography geogB );
```

国国

Returns `true` if two geometries intersect. Geometries intersect if they have any point in common.

For geography, a distance tolerance of 0.00001 meters is used (so points that are very close are considered to intersect).

In mathematical terms: $ST\_Intersects(A, B) \Leftrightarrow A \cap B \neq \emptyset$

Geometries intersect if their DE-9IM Intersection Matrix matches one of:

- T*****
- *T*****
- ***T*****
- ****T****

Spatial intersection is implied by all the other spatial relationship tests, except **ST_Disjoint**, which tests that geometries do NOT intersect.



Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Changed: 3.0.0 SFCGAL version removed and native support for 2D TINS added.

Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION.

Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Performed by the GEOS module (for geometry), geography is native

Availability: 1.5 support for geography was introduced.



Note

For geography, this function has a distance tolerance of about 0.00001 meters and uses the sphere rather than spheroid calculation.



Note

NOTE: this is the "allowable" version that returns a boolean, not an integer.



This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**. s2.1.1.2 //s2.1.13.3 - `ST_Intersects(g1, g2 ) --> Not (ST_Disjoint(g1, g2 ))`



This method implements the SQL/MM specification. SQL-MM 3: 5.1.27



This method supports Circular Strings and Curves.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

图例

```
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
st_intersects
-----
f
(1 row)
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
st_intersects
-----
t
(1 row)

-- Look up in table. Make sure table has a GiST index on geometry column for faster lookup.
SELECT id, name FROM cities WHERE ST_Intersects(geom, 'SRID=4326;POLYGON((28 53,27.707 ↵
52.293,27 52,26.293 52.293,26 53,26.293 53.707,27 54,27.707 53.707,28 53))');
id | name
----+-----
2 | Minsk
(1 row)
```

图例

```
SELECT ST_Intersects(
  'SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568) '::geography,
  'SRID=4326;POINT(-43.23456 72.4567772) '::geography
);

st_intersects
-----
t
```

图例

&&, [ST_3DIntersects](#), [ST_Disjoint](#)

7.11.1.10 ST_LineCrossingDirection

ST_LineCrossingDirection — Returns a number indicating the crossing behavior of two LineStrings

Synopsis

integer **ST_LineCrossingDirection**(geometry linestringA, geometry linestringB);

图例

Given two linestrings returns an integer between -3 and 3 indicating what kind of crossing behavior exists between them. 0 indicates no crossing. This is only supported for LINESTRINGs.

The crossing number has the following meaning:

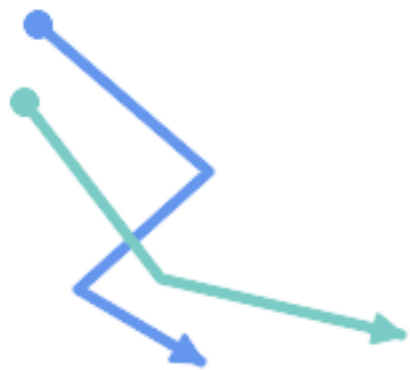
- 0: LINE NO CROSS
- -1: LINE CROSS LEFT

- 1: LINE CROSS RIGHT
- -2: LINE MULTICROSS END LEFT
- 2: LINE MULTICROSS END RIGHT
- -3: LINE MULTICROSS END SAME FIRST LEFT
- 3: LINE MULTICROSS END SAME FIRST RIGHT

Availability: 1.4

图例

Example: LINE CROSS LEFT and LINE CROSS RIGHT

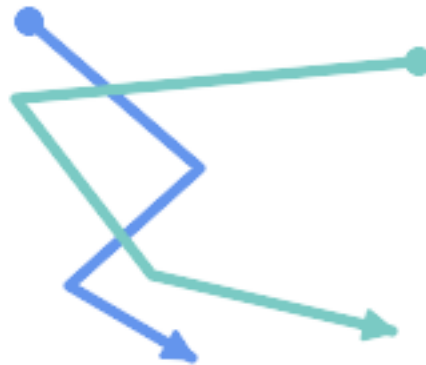


Blue: Line A; Green: Line B

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,  
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A  
FROM (SELECT  
  ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,  
  ST_GeomFromText('LINESTRING (20 140, 71 74, 161 53)') As lineB  
  ) As foo;
```

A_cross_B	B_cross_A
-1	1

Example: LINE MULTICROSS END SAME FIRST LEFT and LINE MULTICROSS END SAME FIRST RIGHT

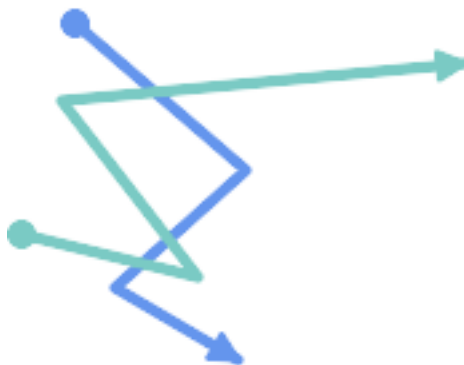


Blue: Line A; Green: Line B

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
      ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
      ST_GeomFromText('LINESTRING(171 154,20 140,71 74,161 53)') As lineB
    ) As foo;
```

A_cross_B	B_cross_A
3	-3

Example: LINE MULTICROSS END LEFT and LINE MULTICROSS END RIGHT



Blue: Line A; Green: Line B

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
      ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
      ST_GeomFromText('LINESTRING(5 90, 71 74, 20 140, 171 154)') As lineB
    ) As foo;
```

A_cross_B	B_cross_A
-2	2

Example: Finds all streets that cross

```
SELECT s1.gid, s2.gid, ST_LineCrossingDirection(s1.geom, s2.geom)
FROM streets s1 CROSS JOIN streets s2
ON (s1.gid != s2.gid AND s1.geom && s2.geom )
WHERE ST_LineCrossingDirection(s1.geom, s2.geom)
> 0;
```

☒☒

ST_Crosses

7.11.1.11 ST_OrderingEquals

ST_OrderingEquals — Tests if two geometries represent the same geometry and have points in the same directional order

Synopsis

boolean **ST_OrderingEquals**(geometry A, geometry B);

☒☒

ST_OrderingEquals compares two geometries and returns t (TRUE) if the geometries are equal and the coordinates are in the same order; otherwise it returns f (FALSE).



Note

This function is implemented as per the ArcSDE SQL specification rather than SQL-MM. http://edndoc.esri.com/arcsgis/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals



This method implements the SQL/MM specification. SQL-MM 3: 5.1.43

☒☒

```
SELECT ST_OrderingEquals(
  'LINESTRING(0 0, 10 10)',
  'LINESTRING(0 0, 5 5, 10 10)');
```

```
st_orderingequals
-----
f
```

```
SELECT ST_OrderingEquals(
```

```
'LINESTRING(0 0, 10 10)',
'LINESTRING(0 0, 10 10)');

st_orderingequals
-----
t

SELECT ST_OrderingEquals(
  'POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))',
  'POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))');

st_orderingequals
-----
f
```

☒☒

&&, [ST_Equals](#), [ST_Reverse](#)

7.11.1.12 ST_Overlaps

ST_Overlaps — Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other

Synopsis

boolean **ST_Overlaps**(geometry A, geometry B);

☒☒

Returns TRUE if geometry A and B "spatially overlap". Two geometries overlap if they have the same dimension, their interiors intersect in that dimension. and each has at least one point inside the other (or equivalently, neither one covers the other). The overlaps relation is symmetric and irreflexive.

In mathematical terms: $ST\_Overlaps(A, B) \Leftrightarrow (dim(A) = dim(B) = dim(Int(A) \cap Int(B))) \wedge (A \not\subset B \wedge B \not\subset A)$



Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Overlaps`.

GEOS 简体中文



Important

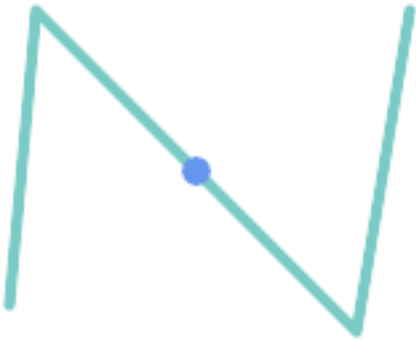
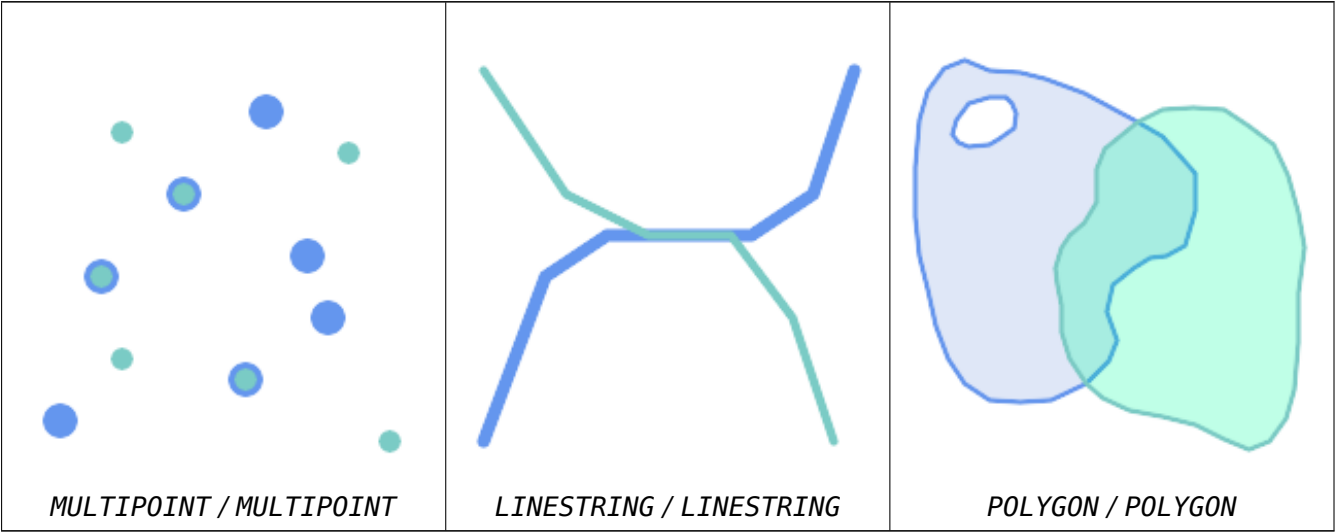
Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

NOTE: this is the "allowable" version that returns a boolean, not an integer.

- ☑ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3](#)
- ☑ This method implements the SQL/MM specification. SQL-MM 3: 5.1.32

☒☒

ST_Overlaps returns TRUE in the following situations:



A Point on a LineString is contained, but since it has lower dimension it does not overlap or cross.

```
SELECT ST_Overlaps(a,b) AS overlaps, ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects, ST_Contains(b,a) AS b_contains_a
FROM (SELECT ST_GeomFromText('POINT (100 100)') As a,
       ST_GeomFromText('LINESTRING (30 50, 40 160, 160 40, 180 160)') AS b) AS t

overlaps | crosses | intersects | b_contains_a
-----+-----+-----+-----
f        | f       | t         | t
```



A LineString that partly covers a Polygon intersects and crosses, but does not overlap since it has different dimension.

```
SELECT ST_Overlaps(a,b) AS overlaps,      ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,  ST_Contains(a,b) AS contains
FROM (SELECT ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
       ST_GeomFromText('LINESTRING(10 10, 190 190)') AS b) AS t;
```

overlap	crosses	intersects	contains
f	t	t	f



Two Polygons that intersect but with neither contained by the other overlap, but do not cross because their intersection has the same dimension.

```
SELECT ST_Overlaps(a,b) AS overlaps,      ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,  ST_Contains(b, a) AS b_contains_a,
       ST_Dimension(a) AS dim_a, ST_Dimension(b) AS dim_b,
       ST_Dimension(ST_Intersection(a,b)) AS dim_int
FROM (SELECT ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
       ST_GeomFromText('POLYGON ((110 180, 20 60, 130 90, 110 180))') AS b) AS t;
```

overlaps	crosses	intersects	b_contains_a	dim_a	dim_b	dim_int
t	f	t	f	2	2	2

[ST_Contains](#), [ST_Crosses](#), [ST_Dimension](#), [ST_Intersects](#)

7.11.1.13 ST_Relate

ST_Relate — Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix

Synopsis

```
boolean ST_Relate(geometry geomA, geometry geomB, text intersectionMatrixPattern);
text ST_Relate(geometry geomA, geometry geomB);
text ST_Relate(geometry geomA, geometry geomB, integer boundaryNodeRule);
```

These functions allow testing and evaluating the spatial (topological) relationship between two geometries, as defined by the [Dimensionally Extended 9-Intersection Model](#) (DE-9IM).

The DE-9IM is specified as a 9-element matrix indicating the dimension of the intersections between the Interior, Boundary and Exterior of two geometries. It is represented by a 9-character text string using the symbols 'F', '0', '1', '2' (e.g. 'FF1FF0102').

A specific kind of spatial relationship can be tested by matching the intersection matrix to an *intersection matrix pattern*. Patterns can include the additional symbols 'T' (meaning "intersection is non-empty") and '*' (meaning "any value"). Common spatial relationships are provided by the named functions [ST_Contains](#), [ST_ContainsProperly](#), [ST_Covers](#), [ST_CoveredBy](#), [ST_Crosses](#), [ST_Disjoint](#), [ST_Equals](#), [ST_Intersects](#), [ST_Overlaps](#), [ST_Touches](#), and [ST_Within](#). Using an explicit pattern allows testing multiple conditions of intersects, crosses, etc in one step. It also allows testing spatial relationships which do not have a named spatial relationship function. For example, the relationship "Interior-Intersects" has the DE-9IM pattern T*****, which is not evaluated by any named predicate.

For more information refer to [Section 5.1](#).

Variant 1: Tests if two geometries are spatially related according to the given intersectionMatrixPattern



Note

Unlike most of the named spatial relationship predicates, this does NOT automatically include an index call. The reason is that some relationships are true for geometries which do NOT intersect (e.g. Disjoint). If you are using a relationship pattern that requires intersection, then include the && index call.



Note

It is better to use a named relationship function if available, since they automatically use a spatial index where one exists. Also, they may implement performance optimizations which are not available with full relate evaluation.

Variant 2: Returns the DE-9IM matrix string for the spatial relationship between the two input geometries. The matrix string can be tested for matching a DE-9IM pattern using [ST_RelateMatch](#).

Variant 3: Like variant 2, but allows specifying a **Boundary Node Rule**. A boundary node rule allows finer control over whether the endpoints of MultiLineStrings are considered to lie in the DE-9IM Interior or Boundary. The boundaryNodeRule values are:

- 1: **OGC-Mod2** - line endpoints are in the Boundary if they occur an odd number of times. This is the rule defined by the OGC SFS standard, and is the default for ST_Relate.
- 2: **Endpoint** - all endpoints are in the Boundary.
- 3: **MultivalentEndpoint** - endpoints are in the Boundary if they occur more than once. In other words, the boundary is all the "attached" or "inner" endpoints (but not the "unattached/outer" ones).
- 4: **MonovalentEndpoint** - endpoints are in the Boundary if they occur only once. In other words, the boundary is all the "unattached" or "outer" endpoints.

This function is not in the OGC spec, but is implied. see s2.1.13.2

✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3

✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.25

GEOS 国国国国国

Enhanced: 2.0.0 - added support for specifying boundary node rule.



Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

国国

Using the boolean-valued function to test spatial relationships.

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer( 'POINT(1 2)', 2), '0FFFFF212');
st_relate
-----
t

SELECT ST_Relate(POINT(1 2)', ST_Buffer( 'POINT(1 2)', 2), '*FF*FF212');
st_relate
-----
t
```

Testing a custom spatial relationship pattern as a query condition, with && to enable using a spatial index.

```
-- Find compounds that properly intersect (not just touch) a poly (Interior Intersects)

SELECT c.* , p.name As poly_name
  FROM polys AS p
 INNER JOIN compounds As c
    ON c.geom && p.geom
    AND ST_Relate(p.geom, c.geom, 'T*****');
```

Computing the intersection matrix for spatial relationships.

```
SELECT ST_Relate( 'POINT(1 2)',
                  ST_Buffer( 'POINT(1 2)', 2));
-----
0FFFFFF212

SELECT ST_Relate( 'LINESTRING(1 2, 3 4)',
                  'LINESTRING(5 6, 7 8)' );
-----
FF1FF0102
```

Using different Boundary Node Rules to compute the spatial relationship between a LineString and a MultiLineString with a duplicate endpoint (3 3):

- Using the **OGC-Mod2** rule (1) the duplicate endpoint is in the **interior** of the MultiLineString, so the DE-9IM matrix entry [aB:bI] is 0 and [aB:bB] is F.
- Using the **Endpoint** rule (2) the duplicate endpoint is in the **boundary** of the MultiLineString, so the DE-9IM matrix entry [aB:bI] is F and [aB:bB] is 0.

```
WITH data AS (SELECT
  'LINESTRING(1 1, 3 3)::geometry AS a_line,
  'MULTILINESTRING((3 3, 3 5), (3 3, 5 3)):: geometry AS b_multiline
)
SELECT ST_Relate( a_line, b_multiline, 1) AS bnr_mod2,
       ST_Relate( a_line, b_multiline, 2) AS bnr_endpoint
FROM data;

bnr_mod2 | bnr_endpoint
-----+-----
FF10F0102 | FF1F00102
```

☒☒

Section 5.1, [ST_RelateMatch](#), [ST_Contains](#), [ST_ContainsProperly](#), [ST_Covers](#), [ST_CoveredBy](#), [ST_Crosses](#), [ST_Disjoint](#), [ST_Equals](#), [ST_Intersects](#), [ST_Overlaps](#), [ST_Touches](#), [ST_Within](#)

7.11.1.14 ST_RelateMatch

`ST_RelateMatch` — Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern

Synopsis

boolean **ST_RelateMatch**(text intersectionMatrix, text intersectionMatrixPattern);

☒☒

Tests if a [Dimensionally Extended 9-Intersection Model](#) (DE-9IM) `intersectionMatrix` value satisfies an `intersectionMatrixPattern`. Intersection matrix values can be computed by [ST_Relate](#).

For more information refer to Section 5.1.

GEOS ☒☒☒☒☒

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

SQL

```
SELECT ST_RelateMatch('101202FFF', 'TTTTTTFFF') ;
-- result --
t
```

Patterns for common spatial relationships matched against intersection matrix values, for a line in various positions relative to a polygon

```
SELECT pat.name AS relationship, pat.val AS pattern,
       mat.name AS position, mat.val AS matrix,
       ST_RelateMatch(mat.val, pat.val) AS match
FROM (VALUES ( 'Equality', 'T1FF1FFF1' ),
      ( 'Overlaps', 'T*T***T**' ),
      ( 'Within', 'T*F**F***' ),
      ( 'Disjoint', 'FF*FF****' )) AS pat(name,val)
CROSS JOIN
  (VALUES ('non-intersecting', 'FF1FF0212'),
    ('overlapping', '1010F0212'),
    ('inside', '1FF0FF212')) AS mat(name,val);
```

relationship	pattern	position	matrix	match
Equality	T1FF1FFF1	non-intersecting	FF1FF0212	f
Equality	T1FF1FFF1	overlapping	1010F0212	f
Equality	T1FF1FFF1	inside	1FF0FF212	f
Overlaps	T*T***T**	non-intersecting	FF1FF0212	f
Overlaps	T*T***T**	overlapping	1010F0212	t
Overlaps	T*T***T**	inside	1FF0FF212	f
Within	T*F**F***	non-intersecting	FF1FF0212	f
Within	T*F**F***	overlapping	1010F0212	f
Within	T*F**F***	inside	1FF0FF212	t
Disjoint	FF*FF****	non-intersecting	FF1FF0212	t
Disjoint	FF*FF****	overlapping	1010F0212	f
Disjoint	FF*FF****	inside	1FF0FF212	f

SQL

Section 5.1, [ST_Relate](#)

7.11.1.15 ST_Touches

ST_Touches — Tests if two geometries have at least one point in common, but their interiors do not intersect

Synopsis

boolean **ST_Touches**(geometry A, geometry B);

SQL

Returns TRUE if A and B intersect, but their interiors do not intersect. Equivalently, A and B have at least one point in common, and the common points lie in at least one boundary. For Point/Point inputs the relationship is always FALSE, since points do not have a boundary.

In mathematical terms: $ST\_Touches(A, B) \Leftrightarrow (Int(A) \cap Int(B) = \emptyset) \wedge (A \cap B \neq \emptyset)$

This relationship holds if the DE-9IM Intersection Matrix for the two geometries matches one of:

- FT*****
- F**T*****
- F***T****



Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid using an index, use `_ST_Touches` instead.



Important

Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`



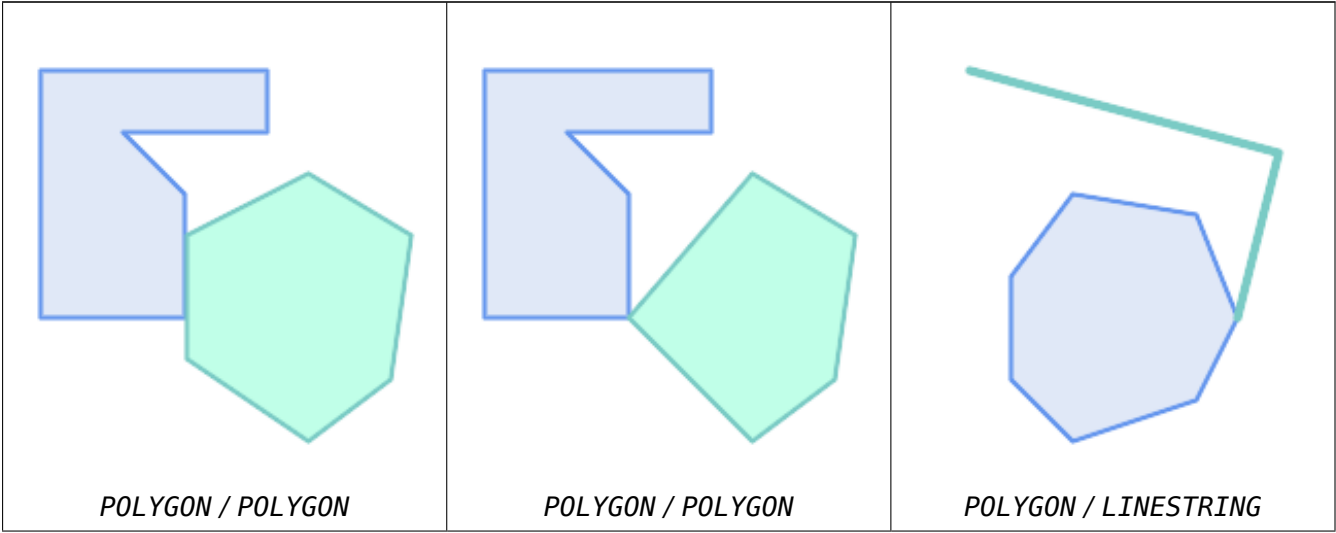
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3](#)

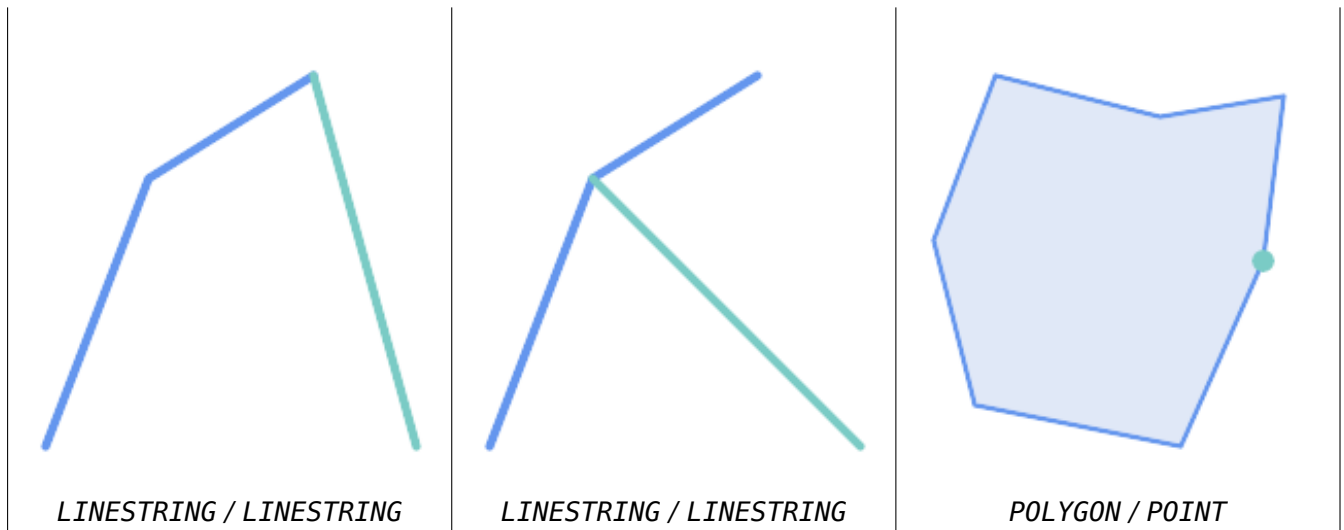


This method implements the SQL/MM specification. SQL-MM 3: 5.1.28



The `ST_Touches` predicate returns TRUE in the following examples.





```
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(1 1)::geometry');
st_touches
-----
f
(1 row)

SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(0 2)::geometry');
st_touches
-----
t
(1 row)
```

7.11.1.16 ST_Within

ST_Within — Tests if every point of A lies in B, and their interiors have a point in common

Synopsis

boolean **ST_Within**(geometry A, geometry B);



Returns TRUE if geometry A is within geometry B. A is within B if and only if all points of A lie inside (i.e. in the interior or boundary of) B (or equivalently, no points of A lie in the exterior of B), and the interiors of A and B have at least one point in common.

For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.

In mathematical terms: $ST\_Within(A, B) \Leftrightarrow (A \sqsubset B = A) \wedge (Int(A) \sqcap Int(B) \neq \emptyset)$

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

```
--a circle within a circle
SELECT ST_Within(smallc,smallc) As smallinsmall,
       ST_Within(smallc, bigc) As smallinbig,
       ST_Within(bigc,smallc) As biginsmall,
       ST_Within(ST_Union(smallc, bigc), bigc) as unioninbig,
       ST_Within(bigc, ST_Union(smallc, bigc)) as beginunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion
FROM
(
SELECT ST_Buffer(ST_GeomFromText('POINT(50 50)'), 20) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(50 50)'), 40) As bigc) As foo;
--Result
smallinsmall | smallinbig | biginsmall | unioninbig | beginunion | bigisunion
-----+-----+-----+-----+-----+-----
t            | t          | f          | t          | t          | t
(1 row)
```



☒☒

[ST_Contains](#), [ST_CoveredBy](#), [ST_Equals](#), [ST_IsValid](#)

7.11.2 Distance Relationships

7.11.2.1 ST_3DDWithin

ST_3DDWithin — Tests if two 3D geometries are within a given 3D distance

Synopsis

boolean **ST_3DDWithin**(geometry g1, geometry g2, double precision distance_of_srid);

☒☒

Returns true if the 3D distance between two geometry values is no larger than distance `distance_of_srid`. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense the source geometries must be in the same coordinate system (have the same SRID).



Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.



This method implements the SQL/MM specification. SQL-MM ?

2.0.0 简体中文

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point  ←
and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same  ←
units as final.
SELECT ST_3DDWithin(
    ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)'),2163),
    ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546  ←
20)'),2163),
    126.8
) As within_dist_3d,
ST_DWithin(
    ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)'),2163),
    ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546  ←
20)'),2163),
    126.8
) As within_dist_2d;

within_dist_3d | within_dist_2d
-----+-----
f              | t
```

[ST_3DDFullyWithin](#), [ST_DWithin](#), [ST_DFullyWithin](#), [ST_3DDistance](#), [ST_Distance](#), [ST_3DMaxDistance](#), [ST_Transform](#)

7.11.2.2 ST_3DDFullyWithin

ST_3DDFullyWithin — Tests if two 3D geometries are entirely within a given 3D distance

Synopsis

boolean **ST_3DDFullyWithin**(geometry g1, geometry g2, double precision distance);

Returns true if the 3D geometries are fully within the specified distance of one another. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.



Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

2.0.0

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.

❏❏

```
-- This compares the difference between fully within and distance within as well
-- as the distance fully within for the 2D footprint of the line/point vs. the 3d fully ←
within
SELECT ST_3DDFullyWithin(geom_a, geom_b, 10) as D3DFullyWithin10, ST_3DDWithin(geom_a, ←
geom_b, 10) as D3DWithin10,
ST_DFullyWithin(geom_a, geom_b, 20) as D2DFullyWithin20,
ST_3DDFullyWithin(geom_a, geom_b, 20) as D3DFullyWithin20 from
(select ST_GeomFromEWKT('POINT(1 1 2)') as geom_a,
ST_GeomFromEWKT('LINESTRING(1 5 2, 2 7 20, 1 9 100, 14 12 3)') as geom_b) t1;
d3dfullywithin10 | d3dwithin10 | d2dfullywithin20 | d3dfullywithin20
-----+-----+-----+-----
f              | t          | t          | f
```

❏❏

ST_3DDWithin, ST_DWithin, ST_DFullyWithin, ST_3DMaxDistance

7.11.2.3 ST_DFullyWithin

ST_DFullyWithin — Tests if a geometry is entirely inside a distance of another

Synopsis

boolean **ST_DFullyWithin**(geometry g1, geometry g2, double precision distance);

❏❏

Returns true if g2 is entirely within distance of g1. Visually, the condition is true if g2 is contained within a distance buffer of g1. The distance is specified in units defined by the spatial reference system of the geometries.



Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

1.5.0 简体中文.

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

表头

[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);
```

表头

Returns true if the geometries are within a given distance

For geometry: The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must be in the same coordinate system (have the same SRID).

For geography: units are in meters and distance measurement defaults to `use_spheroid = true`. For faster evaluation use `use_spheroid = false` to measure on the sphere.



Note

Use [ST_3DDWithin](#) for 3D geometries.



Note

This function call includes a bounding box comparison that makes use of any indexes that are available on the geometries.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).

Availability: 1.5.0 support for geography was introduced

Enhanced: 2.1.0 improved speed for geography. See [Making Geography faster](#) for details.

Enhanced: 2.1.0 support for curved geometries was introduced.

Prior to 1.3, [ST_Expand](#) was commonly used in conjunction with `&&` and `ST_Distance` to test for distance, and in pre-1.3.4 this function used that logic. From 1.3.4, `ST_DWithin` uses a faster short-circuit distance function.

❏

```
-- Find the nearest hospital to each school
-- that is within 3000 units of the school.
-- We do an ST_DWithin search to utilize indexes to limit our search list
-- that the non-indexable ST_Distance needs to process
-- If the units of the spatial reference is meters then units would be meters
SELECT DISTINCT ON (s.gid) s.gid, s.school_name, s.geom, h.hospital_name
FROM schools s
LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
ORDER BY s.gid, ST_Distance(s.geom, h.geom);

-- The schools with no close hospitals
-- Find all schools with no hospital within 3000 units
-- away from the school. Units is in units of spatial ref (e.g. meters, feet, degrees)
SELECT s.gid, s.school_name
FROM schools s
LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
WHERE h.gid IS NULL;

-- Find broadcasting towers that receiver with limited range can receive.
-- Data is geometry in Spherical Mercator (SRID=3857), ranges are approximate.

-- Create geometry index that will check proximity limit of user to tower
CREATE INDEX ON broadcasting_towers using gist (geom);

-- Create geometry index that will check proximity limit of tower to user
CREATE INDEX ON broadcasting_towers using gist (ST_Expand(geom, sending_range));

-- Query towers that 4-kilometer receiver in Minsk Hackerspace can get
-- Note: two conditions, because shorter LEAST(b.sending_range, 4000) will not use index.
SELECT b.tower_id, b.geom
FROM broadcasting_towers b
WHERE ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', 4000)
AND ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', b.sending_range);
```

❏

ST_Distance, ST_3DDWithin

7.11.2.5 ST_PointInsideCircle

ST_PointInsideCircle — Tests if a point geometry is inside a circle defined by a center and radius

Synopsis

boolean **ST_PointInsideCircle**(geometry a_point, float center_x, float center_y, float radius);

❏

Returns true if the geometry is a point and is inside the circle with center center_x,center_y and radius radius.



Warning

Does not use spatial indexes. Use **ST_DWithin** instead.

Availability: 1.2

Changed: 2.2.0 In prior versions this was called ST_Point_Inside_Circle


```
SELECT ST_PointInsideCircle(ST_Point(1,2), 0.5, 2, 3);
 st_pointinsidecircle
-----
t
```


ST DWithin

7.12 Measurement Functions

7.12.1 ST_Area

ST Area — □□□□□□□□□□□□□□.

Synopsis

```
float ST_Area(geometry g1);  
float ST_Area(geography geog, boolean use_spheroid = true);
```


ST_Surface(*geom*) - ST_MultiSurface(*geom*) - *geom*. *geom*, SRID *geom* 2 *geom*. *geom*, *geom* (curved surface) *geom*. *geom*, ST_Area(*geom*,false) *geom*.

2.0.0 2 (polyhedral surface)

GeographicLib 2.2.0, Proj 4.9.0.

Changed: 3.0.0 - does not depend on SFCGAL anymore.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 8.1.2, 9.5.3



This function supports Polyhedral surfaces.



Note

XXXXXXXXXXXX, (2.5 XXXXXXX) 2 XXXXXXXXXXXXXXXXXXXX. 2.5 XXXXXXX 0 XXXX (non-zero) XXXXXXXXXXXXXXXX, XY XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.



 (plot) . EPSG:2249 .

[illegible][illegible]

```

select ST_Area(geom) sqft,
       ST_Area(ST_Transform(geom, 26986)) As sqm
from (
    select
        'SRID=2249;POLYGON((743238 2967416,743238 2967450,
        743265 2967450,743265.625 2967416,743238 2967416))' :: geometry geom
    ) subquery;

```

本圖係根據 1984 年 1 月 1 日之 WGS84 4326 座標系統繪製。圖中所有之點位均係根據該座標系統計算所得。圖中所有之點位均係根據該座標系統計算所得。圖中所有之點位均係根據該座標系統計算所得。

```
select ST_Area(geog) / 0.3048 ^ 2 sqft_spheroid,
       ST_Area(geog, false) / 0.3048 ^ 2 sqft_sphere,
       ST_Area(geog) sqm_spheroid
from (
```

```

select ST_Transform(
  'SRID=2249;POLYGON((743238 2967416,743238 2967450,743265
    2967450,743265.625 2967416,743238 2967416))'::geometry,
    4326
  ) :: geography geog
) as subquery;

```

If your data is in geography already:

```
select ST_Area(geog) / 0.3048 ^ 2 sqft,
       ST_Area(the_geog) sqm
from somegeogtable;
```



ST 3DArea, ST_GeomFromEWKT, ST_LengthSpheroid, ST_Perimeter, ST_Transform

7.12.2 ST Azimuth

ST Azimuth — □□□□□□ 2 □□□□□□□□□□.

Synopsis

```
float ST_Azimuth(geometry origin, geometry target);  
float ST_Azimuth(geography origin, geography target);
```


Returns the azimuth in radians of the target point from the origin point, or NULL if the two points are coincident. The azimuth angle is a positive clockwise angle referenced from the positive Y axis (geometry) or the North meridian (geography): North = 0; Northeast = $\pi/4$; East = $\pi/2$; Southeast = $3\pi/4$; South = π ; Southwest $5\pi/4$; West = $3\pi/2$; Northwest = $7\pi/4$.

For the geography type, the azimuth solution is known as the **inverse geodesic problem**.

The azimuth is a mathematical concept defined as the angle between a reference vector and a point, with angular units in radians. The result value in radians can be converted to degrees using the PostgreSQL function `degrees()`.

ST_Translate 函数用于将几何体沿指定的 X 和 Y 轴偏移指定的距离。有关更多信息，请参阅 [Plpgsqlfunctions PostgreSQL wiki section](#) 中的 upgis_lineshift 函数。

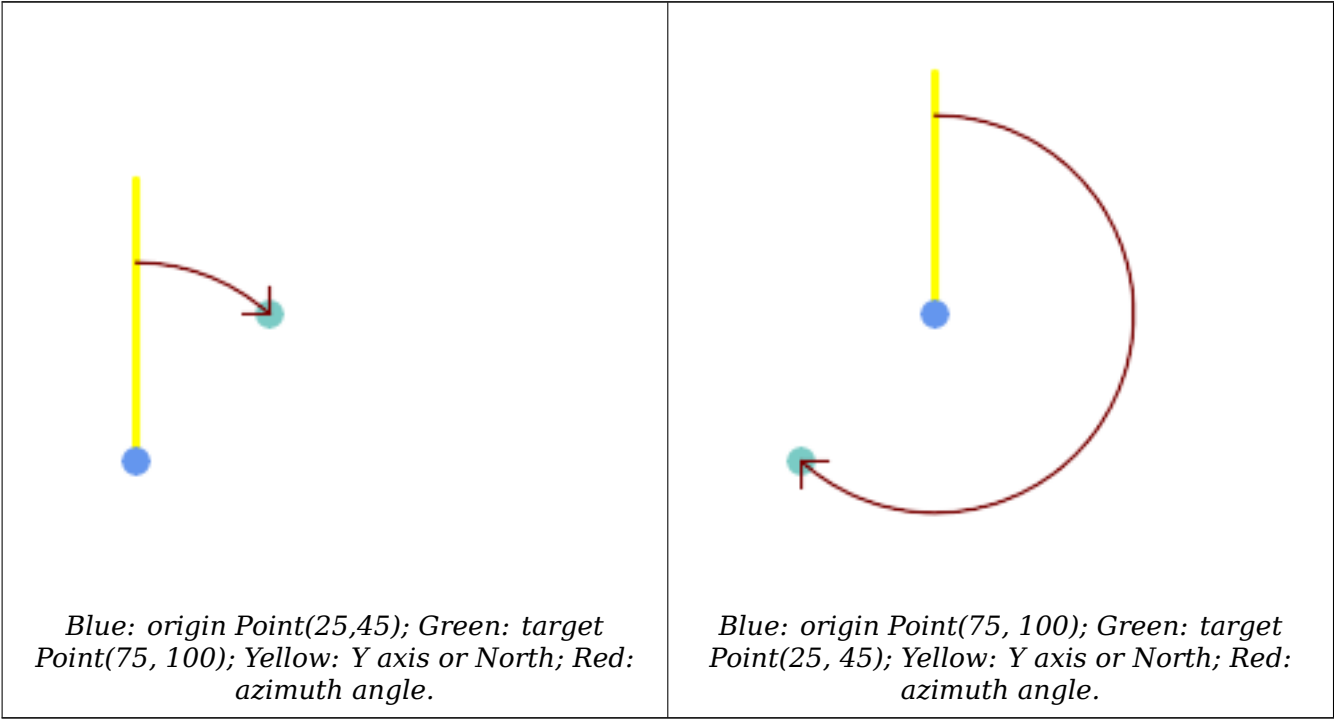
- 1.1.0 版本。
- 2.0.0 版本。
- 2.2.0 版本。GeographicLib 4.9.0 版本。

SQL

SQL 查询

```
SELECT degrees(ST_Azimuth( ST_Point(25, 45), ST_Point(75, 100))) AS degA_B,
       degrees(ST_Azimuth( ST_Point(75, 100), ST_Point(25, 45) )) AS degB_A;
```

degA_b	degB_a
42.2736890060937	222.273689006094



SQL

[ST_Angle](#), [ST_Translate](#), [ST_Project](#), [PostgreSQL Math Functions](#)

7.12.3 ST_Angle

ST_Angle — 返回由 3 个点 (longest) 定义的角。

Synopsis

float **ST_Angle**(geometry point1, geometry point2, geometry point3, geometry point4);
float **ST_Angle**(geometry line1, geometry line2);

返回

返回 3 个角度 (longest) 的度数。

Variant 1: computes the angle enclosed by the points P1-P2-P3. If a 4th point provided computes the angle points P1-P2 and P3-P4

Variant 2: computes the angle between two vectors S1-E1 and S2-E2, defined by the start and end points of the input lines

PostgreSQL 的 `degrees()` 函数返回度数。PostgreSQL 的 `degrees()` 函数返回度数。

Note that `ST_Angle(P1,P2,P3) = ST_Angle(P2,P1,P2,P3)`.

Availability: 2.5.0

示例

计算三个点的角度

```
SELECT degrees( ST_Angle('POINT(0 0)', 'POINT(10 10)', 'POINT(20 0)') );
```

```
degrees
```

```
-----  
270
```

Angle between vectors defined by four points

```
SELECT degrees( ST_Angle('POINT (10 10)', 'POINT (0 0)', 'POINT(90 90)', 'POINT (100 80)') ) ←  
);
```

```
degrees
```

```
-----  
269.99999999999999
```

Angle between vectors defined by the start and end points of lines

```
SELECT degrees( ST_Angle('LINESTRING(0 0, 0.3 0.7, 1 1)', 'LINESTRING(0 0, 0.2 0.5, 1 0)') ) ←  
);
```

```
degrees
```

```
-----  
45
```

函数

ST_Azimuth

7.12.4 ST_ClosestPoint

ST_ClosestPoint — Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.

Synopsis

geometry **ST_ClosestPoint**(geometry geom1, geometry geom2);
geography **ST_ClosestPoint**(geography geom1, geography geom2, boolean use_spheroid = true);

返回

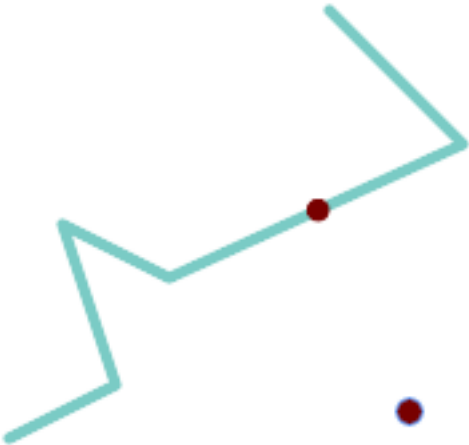
Returns the 2-dimensional point on geom1 that is closest to geom2. This is the first point of the shortest line between the geometries (as computed by [ST_ShortestLine](#)).



Note
3 返回 [ST_3DClosestPoint](#) 返回。

Enhanced: 3.4.0 - Support for geography.
1.5.0 返回。

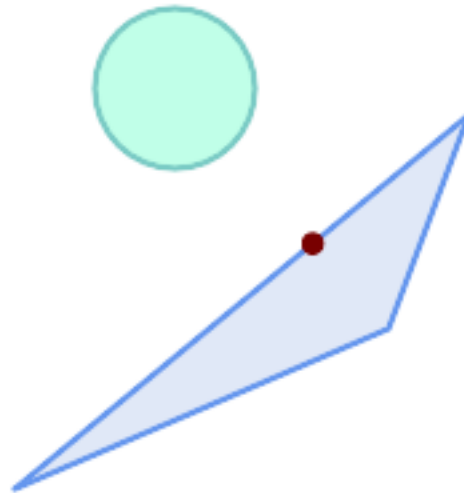
返回



The closest point for a Point and a LineString is the point itself. The closest point for a LineString and a Point is a point on the line.

```
SELECT ST_AsText( ST_ClosestPoint(pt,line)) AS cp_pt_line,
       ST_AsText( ST_ClosestPoint(line,pt)) AS cp_line_pt
FROM (SELECT 'POINT (160 40)::geometry AS pt,
            'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)::geometry AS line ) AS t;
```

cp_pt_line	cp_line_pt
POINT(160 40)	POINT(125.75342465753425 115.34246575342466)



The closest point on polygon A to polygon B

```
SELECT ST_AsText( ST_ClosestPoint(
                        'POLYGON ((190 150, 20 10, 160 70, 190 150))',
                        ST_Buffer('POINT(80 160)', 30)
                    )) As ptwkt;
-----
POINT(131.59149149528952 101.89887534906197)
```



ST 3DClosestPoint, ST Distance, ST LongestLine, ST ShortestLine, ST MaxDistance

7.12.5 ST_3DClosestPoint

ST_3DClosestPoint — g2 g1 3 . 3D .

Synopsis

```
geometry ST_3DClosestPoint(geometry g1, geometry g2);
```

[illegible]

- ✔ This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

Figure 2.2.0: 2D plot of $\rho(x, y, z)$, ($\rho(x, y, z) \geq 0$ for all x, y, z) 2D plot of $\rho(x, y, z)$. 2D plot of $\rho(x, y, z)$, ($\rho(x, y, z) \geq 0$ for all x, y, z).



ST_3DClosestPoint -- 3D, 2D 最近点 (closest point)

```
SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,
       ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt
FROM (SELECT 'POINT(100 100 30)::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)::geometry As line
      ) As foo;
```

cp3d_line_pt		
cp2d_line_pt		

POINT(54.6993798867619 128.935022917228 11.5475869506606)		POINT(73.0769230769231 115.384615384615)

ST_3DClosestPoint -- 3D, 2D 最近点 (closest point)

```
SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,
       ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)::geometry As line
      ) As foo;
```

cp3d_line_pt		cp2d_line_pt
-----+		

POINT(54.6993798867619 128.935022917228 11.5475869506606)		POINT(50 75)

ST_3DClosestPoint -- 3D, 2D 最近点 (closest point)

```
SELECT ST_AsEWKT(ST_3DClosestPoint(poly, mline)) As cp3d,
       ST_AsEWKT(ST_ClosestPoint(poly, mline)) As cp2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, 100 100 5, 175 150 5))') As poly,
            ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 100 1, 175 155 1), (1 10 2, 5 20 1))') As mline ) As foo;
```

cp3d		cp2d
-----+		
POINT(39.993580415989 54.1889925532825 5)		POINT(20 40)

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);
```

返回

浮点数字, 表示 3 维笛卡尔坐标系 (SRS 4326) 中的距离。

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 支持圆字符串和曲线。支持 3 维笛卡尔坐标系 (SRS 4326) 中的距离。

Enhanced: 2.1.0 improved speed for geography. See [Making Geography faster](#) for details.

修复: 2.1.0 修复了某些情况下的精度问题。

修复: 2.2.0 修复了某些情况下的精度问题。GeographicLib 修复。修复了 Proj 4.9.0 中的某些问题。

Changed: 3.0.0 - does not depend on SFCGAL anymore.

示例

Geometry example - units in planar degrees 4326 is WGS 84 long lat, units are degrees.

```
SELECT ST_Distance(
  'SRID=4326;POINT(-72.1235 42.3521)::geometry',
  'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry ');
-----
0.00150567726382282
```

Geometry example - units in meters (SRID: 3857, proportional to pixels on popular web maps). Although the value is off, nearby ones can be compared correctly, which makes it a good choice for algorithms like KNN or KMeans.

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry', 3857),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry', 3857) ) ←
;
-----
167.441410065196
```

Geometry example - units in meters (SRID: 3857 as above, but corrected by cos(lat) to account for distortion)

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry', 3857),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry', 3857)
) * cosd(42.3521);
-----
123.742351254151
```

Geometry example - units in meters (SRID: 26986 Massachusetts state plane meters) (most accurate for Massachusetts)

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry, 26986),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry, 26986) ) ←
  ;
-----
123.797937878454
```

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (least accurate)

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry, 2163),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry, 2163) ) ←
  ;
-----
126.664256056812
```

地理距离

Same as geometry example but note units in meters - use sphere for slightly faster and less accurate computation.

```
SELECT ST_Distance(gg1, gg2) As spheroid_dist, ST_Distance(gg1, gg2, false) As sphere_dist
FROM (SELECT
  'SRID=4326;POINT(-72.1235 42.3521)::geography as gg1,
  'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geography as gg2
) As foo ;

spheroid_dist | sphere_dist
-----+-----
123.802076746848 | 123.475736916397
```

地理距离

[ST_3DDistance](#), [ST_DWithin](#), [ST_DistanceSphere](#), [ST_DistanceSpheroid](#), [ST_MaxDistance](#), [ST_HausdorffDistance](#), [ST_FrechetDistance](#), [ST_Transform](#)

7.12.7 ST_3DDistance

ST_3DDistance — 计算两个几何体之间的 3D 距离 (SRS 无关) 3 个参数。返回浮点值。

Synopsis

float **ST_3DDistance**(geometry g1, geometry g2);

参数

几何体 g1, 几何体 g2 3 个参数。返回浮点值 (SRS 无关)。



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 函数参考.
- 更新: 2.2.0 版本, 2D 和 3D 函数都支持 Z 坐标 Z 为 0 的函数.
- Changed: 3.0.0 - SFCGAL version removed

函数

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same units as final.
SELECT ST_3DDistance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521 4) '::geometry,2163),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 20) '::geometry,2163)
) As dist_3d,
ST_Distance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521) '::geometry,2163),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546) '::geometry,2163)
) As dist_2d;

dist_3d      |      dist_2d
-----+-----
127.295059324629 | 126.66425605671

-- Multilinestring and polygon both 3d and 2d distance
-- Same example as 3D closest point example
SELECT ST_3DDistance(poly, mline) As dist3d,
    ST_Distance(poly, mline) As dist2d
    FROM (SELECT 'POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, 100 100 5, 175 150 5) ' :::geometry as poly,
        'MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 100 1, 175 155 1), (1 10 2, 5 20 1))' :::geometry as mline) as foo;

dist3d      | dist2d
-----+-----
0.716635696066337 | 0
```

函数

[ST_Distance](#), [ST_3DClosestPoint](#), [ST_3DDWithin](#), [ST_3DMaxDistance](#), [ST_3DShortestLine](#), [ST_Transform](#)

7.12.8 ST_DistanceSphere

ST_DistanceSphere — 计算两个几何体之间的球面距离。PostGIS 1.5 版本引入。

Synopsis

float **ST_DistanceSphere**(geometry geom1lonlatA, geometry geom1lonlatB, float8 radius=6371008);

图

图 2 展示了使用 `ST_DistanceSpheroid` 函数。SRID 4326 是 WGS 1984 地理坐标系。PostGIS 1.5 之前，`ST_DistanceSpheroid` 函数是不可用的。

1.5 版本之前，使用 `ST_DistanceSphere` 函数。1.5 版本之后，使用 `ST_DistanceSpheroid` 函数。

图 2.2.0 展示了 `ST_DistanceSphere` 函数的使用。

图

```
SELECT round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38)', 4326)) As numeric),2) As dist_meters,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(geom),32611),
ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) As dist_utm11_meters,
round(CAST(ST_Distance(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38)', 4326)) As numeric),5) As dist_degrees,
round(CAST(ST_Distance(ST_Transform(geom,32611),
ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) As min_dist_line_point_meters
FROM
(SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As geom)
as foo;
dist_meters | dist_utm11_meters | dist_degrees | min_dist_line_point_meters
-----+-----+-----+-----
70424.47 | 70438.00 | 0.72900 | 65871.18
```

图

`ST_Distance`, `ST_DistanceSpheroid`

7.12.9 ST_DistanceSpheroid

`ST_DistanceSpheroid` — 计算两个几何体之间的球面距离。PostGIS 1.5 之前，使用 `ST_DistanceSphere` 函数。

Synopsis

float **ST_DistanceSpheroid**(geometry geomlonlatA, geometry geomlonlatB, spheroid measurement_spheroid)

图

图 展示了使用 `ST_DistanceSpheroid` 函数。图 展示了使用 `ST_LengthSpheroid` 函数。PostGIS 1.5 之前，使用 `ST_LengthSphere` 函数。



Note

图 展示了使用 SRID 4326 的几何体。图 展示了使用 SRID 4326 的几何体。

1.5 使用 `ST_DistanceSpheroid` 函数。1.5 使用 `ST_DistanceSphere` 函数。

2.2.0 使用 `ST_Distance_Spheroid` 函数。

SQL

```
SELECT round(CAST(
    ST_DistanceSpheroid(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38) ←
    ',4326), 'SPHEROID["WGS 84",6378137,298.257223563]')
    As numeric),2) As dist_meters_spheroid,
    round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 ←
    38)',4326)) As numeric),2) As dist_meters_sphere,
    round(CAST(ST_Distance(ST_Transform(ST_Centroid(geom),32611),
    ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) ←
    As dist_utm11_meters
FROM
    (SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As geom) ←
    as foo;
dist_meters_spheroid | dist_meters_sphere | dist_utm11_meters
-----+-----+-----
70454.92 | 70424.47 | 70438.00
```

SQL

[ST_Distance](#), [ST_DistanceSphere](#)

7.12.10 ST_FrechetDistance

`ST_FrechetDistance` — 计算 3 个最短的 Fréchet 距离。

Synopsis

`float ST_FrechetDistance(geometry g1, geometry g2, float densifyFrac = -1);`

SQL

Implements algorithm for computing the Fréchet distance restricted to discrete points for both geometries, based on [Computing Discrete Fréchet Distance](#). The Fréchet distance is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. Therefore it is often better than the Hausdorff distance.

When the optional `densifyFrac` is specified, this function performs a segment densification before computing the discrete Fréchet distance. The `densifyFrac` parameter sets the fraction by which to densify each segment. Each segment will be split into a number of equal-length subsegments, whose fraction of the total length is closest to the given fraction.

Units are in the units of the spatial reference system of the geometries.



Note

使用 `ST_FrechetDistance` 函数。使用 `ST_FrechetDistance` 函数。

**Note**

The smaller `densifyFrac` we specify, the more accurate Fréchet distance we get. But, the computation time and the memory usage increase with the square of the number of subsegments.

GEOS 函数

Availability: 2.4.0 - requires GEOS >= 3.7.0

SQL

```
postgres=# SELECT st_frechetdistance('LINESTRING (0 0, 100 0)::geometry', 'LINESTRING (0 0, ↵
50 50, 100 0)::geometry');
 st_frechetdistance
-----
70.7106781186548
(1 row)
```

```
SELECT st_frechetdistance('LINESTRING (0 0, 100 0)::geometry', 'LINESTRING (0 0, 50 50, 100 ↵
0)::geometry', 0.5);
 st_frechetdistance
-----
50
(1 row)
```

SQL

ST_HausdorffDistance

7.12.11 ST_HausdorffDistance

`ST_HausdorffDistance` — 返回两个几何体的最短距离 (shortest) 距离。

Synopsis

```
float ST_HausdorffDistance(geometry g1, geometry g2);
float ST_HausdorffDistance(geometry g1, geometry g2, float densifyFrac);
```

SQL

Returns the **Hausdorff distance** between two geometries. The Hausdorff distance is a measure of how similar or dissimilar 2 geometries are.

The function actually computes the "Discrete Hausdorff Distance". This is the Hausdorff distance computed at discrete points on the geometries. The *densifyFrac* parameter can be specified, to provide a more accurate answer by densifying segments before computing the discrete Hausdorff distance. Each segment is split into a number of equal-length subsegments whose fraction of the segment length is closest to the given fraction.

Units are in the units of the spatial reference system of the geometries.

Note

This algorithm is NOT equivalent to the standard Hausdorff distance. However, it computes an approximation that is correct for a large subset of useful cases. One important case is Linestrings that are roughly parallel to each other, and roughly equal in length. This is a useful metric for line matching.

1.5.0



Hausdorff distance (red) and distance (yellow) between two lines

```
SELECT ST_HausdorffDistance(geomA, geomB),
       ST_Distance(geomA, geomB)
FROM (SELECT 'LINESTRING (20 70, 70 60, 110 70, 170 70)'::geometry AS geomA,
            'LINESTRING (20 90, 130 90, 60 100, 190 100)'::geometry AS geomB) AS t;
st_hausdorffdistance | st_distance
-----+-----
37.26206567625497 | 20
```

Example: Hausdorff distance with densification.

```
SELECT ST_HausdorffDistance(
    'LINESTRING (130 0, 0 0, 0 150)'::geometry,
    'LINESTRING (10 10, 10 150, 130 10)'::geometry,
    0.5);
-----
70
```

Example: For each building, find the parcel that best represents it. First we require that the parcel intersect with the building geometry. `DISTINCT ON` guarantees we get each building listed only once. `ORDER BY .. ST_HausdorffDistance` selects the parcel that is most similar to the building.

```
SELECT DISTINCT ON (buildings.gid) buildings.gid, parcels.parcel_id
FROM buildings
  INNER JOIN parcels
    ON ST_Intersects(buildings.geom, parcels.geom)
ORDER BY buildings.gid, ST_HausdorffDistance(buildings.geom, parcels.geom);
```

几何

ST_FrechetDistance

7.12.12 ST_Length

ST_Length — 返回几何对象的长度。

Synopsis

```
float ST_Length(geometry a_2dlinestring);  
float ST_Length(geography geog, boolean use_spheroid = true);
```

几何

几何类型: 点, 线串, **ST_Curve**, **ST_MultiCurve** 返回 2 维几何对象的长度。返回 0 表示空几何。对于 **ST_Perimeter** 函数, 返回几何对象的周长。对于 **ST_Length**, 返回几何对象的长度。

For geography types: computation is performed using the inverse geodesic calculation. Units of length are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If `use_spheroid = false`, then the calculation is based on a sphere instead of a spheroid.

对于 2D 几何类型, **ST_Length2D** 函数返回几何对象的长度。



Warning

注意: 2.0.0 版本中, **ST_Length** 函数对于 geography 类型使用 WGS84 椭球体进行计算。2.0.0 版本中, **ST_Length** 函数对于 geometry 类型使用平面几何进行计算。2.0.0 版本中, **ST_Length** 函数对于 geometry 类型返回 0 表示空几何。对于 **ST_Perimeter** 函数, 返回几何对象的周长。



Note

对于 geography 类型, **ST_Length** 函数使用 WGS84 椭球体进行计算。对于 geometry 类型, **ST_Length** 函数使用平面几何进行计算。对于 geometry 类型, **ST_Length** 函数返回 0 表示空几何。对于 **ST_Perimeter** 函数, 返回几何对象的周长。



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** 函数返回 0 表示空几何。

几何

对于 geography 类型, **ST_Length** 函数使用 WGS84 椭球体进行计算。对于 geometry 类型, **ST_Length** 函数使用平面几何进行计算。对于 geometry 类型, **ST_Length** 函数返回 0 表示空几何。对于 **ST_Perimeter** 函数, 返回几何对象的周长。

```

SELECT ST_Length(ST_GeomFromText('LINESTRING(743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416)',2249));

st_length
-----
122.630744000095

--Transforming WGS 84 LineString to Massachusetts state plane meters
SELECT ST_Length(
    ST_Transform(
        ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, ↔
        -72.123 42.1546)'),
        26986
    )
);

st_length
-----
34309.4563576191

```

图 7.12.12

WGS84 地理坐标转换为平面坐标。

```

-- the default calculation uses a spheroid
SELECT ST_Length(the_geog) As length_spheroid, ST_Length(the_geog,false) As length_sphere
FROM (SELECT ST_GeographyFromText(
'SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, -72.123 42.1546)') As the_geog)
As foo;

length_spheroid | length_sphere
-----+-----
34310.5703627288 | 34346.2060960742

```

图 7.12.13

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。与 ST_Length 类似。

Synopsis

float **ST_Length2D**(geometry a_2dlinestring);

图 7.12.14

图 7.12.14 显示了 ST_Length2D 与 ST_Length 的区别。

函数

ST_Length, ST_3DLength

7.12.14 ST_3DLength

ST_3DLength — 返回 3D 线段的长度。

Synopsis

```
float ST_3DLength(geometry a_3dlinestring);
```

函数

返回 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。 (ST_Length 与 ST_Length2D 函数类似)。



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 函数 ST_Length3D 函数类似。

函数

3D 线段的长度。EPSG:2249 坐标系。EPSG:2249 坐标系。EPSG:2249 坐标系。

```
SELECT ST_3DLength(ST_GeomFromText('LINESTRING(743238 2967416 1,743238 2967450 1,743265
2967450 3,
743265.625 2967416 3,743238 2967416 3)',2249));
ST_3DLength
-----
122.704716741457
```

函数

ST_Length, ST_Length2D

7.12.15 ST_LengthSpheroid

ST_LengthSpheroid — 返回球面线段的长度。

Synopsis

```
float ST_LengthSpheroid(geometry a_geometry, spheroid a_spheroid);
```


7.12.16 ST_LongestLine

ST_LongestLine — 返回 3 个 (longest) 结果。

Synopsis

geometry **ST_LongestLine**(geometry g1, geometry g2);

返回

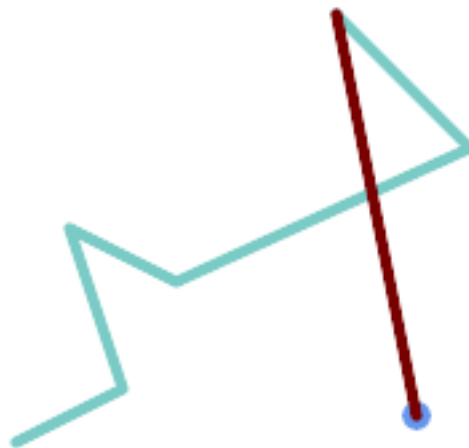
Returns the 2-dimensional longest line between the points of two geometries. The line returned starts on g1 and ends on g2.

The longest line always occurs between two vertices. The function returns the first longest line if more than one is found. The length of the line is equal to the distance returned by [ST_MaxDistance](#).

If g1 and g2 are the same geometry, returns the line between the two vertices farthest apart in the geometry. The endpoints of the line lie on the circle computed by [ST_MinimumBoundingCircle](#).

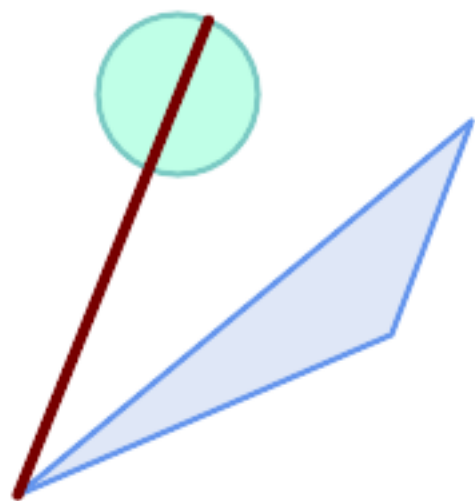
1.5.0 引入。

返回



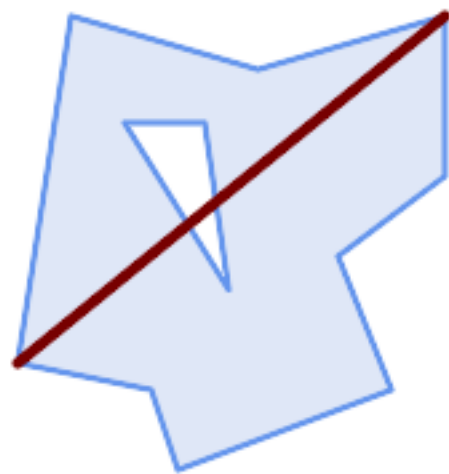
示例

```
SELECT ST_AsText( ST_LongestLine(
    'POINT (160 40)',
    'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)' )
    ) AS lline;
-----
LINESTRING(160 40,130 190)
```



🔍🔍🔍🔍🔍🔍🔍🔍🔍🔍🔍🔍🔍🔍

```
SELECT ST_AsText( ST_LongestLine(
    'POLYGON ((190 150, 20 10, 160 70, 190 150))',
    ST_Buffer('POINT(80 160)', 30)
) ) AS llinewkt;
-----
LINESTRING(20 10,105.3073372946034 186.95518130045156)
```



Longest line across a single geometry. The length of the line is equal to the Maximum Distance. The endpoints of the line lie on the Minimum Bounding Circle.

```
SELECT ST_AsText( ST_LongestLine( geom, geom)) AS llinewkt,
       ST_MaxDistance( geom, geom) AS max_dist,
       ST_Length( ST_LongestLine(geom, geom)) AS lenll
FROM (SELECT 'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 50, 40 180),
       (60 140, 99 77.5, 90 140, 60 140))'::geometry AS geom) AS t;
```

llinewkt	max_dist	lenll
LINESTRING(20 50,180 180)	206.15528128088303	206.15528128088303

3D, 2D 最长线

```
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 最长线

```
SELECT ST_AsEWKT(ST_3DLongestLine(poly, mline)) As lol3d,
       ST_AsEWKT(ST_LongestLine(poly, mline)) As lol2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,
100 100 5, 175 150 5))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125
100 1, 175 155 1),
       (1 10 2, 5 20 1))') As mline ) As foo;

lol3d      |      lol2d
-----+-----
LINESTRING(175 150 5,1 10 2) | LINESTRING(175 150,1 10)
```

ST_3DClosestPoint, ST_3DDistance, ST_LongestLine, ST_3DShortestLine, ST_3DMaxDistance

7.12.18 ST_MaxDistance

ST_MaxDistance — 返回两个几何体之间的最大距离。

Synopsis

float **ST_MaxDistance**(geometry g1, geometry g2);

3D

返回两个几何体之间的最大距离。g1 和 g2 可以是点、线、面。如果两个几何体都是点，则返回它们之间的欧几里得距离。

如果两个几何体都是线，则返回它们之间的最大距离。如果两个几何体都是面，则返回它们之间的最大距离。

1.5.0 版本引入。

3D

3D 最长线

```
SELECT ST_MaxDistance('POINT(0 0)::geometry', 'LINESTRING ( 2 0, 0 2 )::geometry');
-----
2

SELECT ST_MaxDistance('POINT(0 0)::geometry', 'LINESTRING ( 2 2, 2 2 )::geometry');
-----
2.82842712474619
```

Maximum distance between vertices of a single geometry.

```
SELECT ST_MaxDistance('POLYGON ((10 10, 10 0, 0 0, 10 10))'::geometry,  
                      'POLYGON ((10 10, 10 0, 0 0, 10 10))'::geometry);  
-----  
14.142135623730951
```



ST Distance, ST LongestLine, ST DFullyWithin

7.12.19 ST_3DMaxDistance

ST_3DMaxDistance — (SRS) 3

Synopsis

```
float ST_3DMaxDistance(geometry g1, geometry g2);
```



□□□□□□□□, □□□□□□□ 3 □□□□□□□□□□□□□□□□□□□□□□□□ (SRS □□) □□□□□□□.

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

OpenGL: 2.2.0, 2D 3D Z Z 0


```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same units as final.
SELECT ST_3DMaxDistance(
    ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 10000)'),2163),
    ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 20)'),2163)
) As dist_3d,
ST_MaxDistance(
```

```

        ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 10000)'),2163),
        ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 20)'),2163)
    ) As dist_2d;

    dist_3d | dist_2d
-----+-----
24383.7467488441 | 22247.8472107251

```


ST Distance, ST 3DDWithin, ST 3DMaxDistance, ST Transform.

7.12.20 ST_MinimumClearance

ST_MinimumClearance — □□□□□□ (robustness) □□□□□□□□□□ (clearance) □□□□□□.

Synopsis

```
float ST_MinimumClearance(geometry g);
```



(0x00000000) ST_IsValid 00 (0x00000000) ST_IsSimple 000000000000000000000000, 0
00000000 (WKT, KML, GML GeoJSON 0) 0000000000000000000000000000000000
(MapInfo TAB 0) 000000000000000000000000, 000000000000000000000000
00000000000000000000.

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.

[illegible]

- [illegible]

ST Minimum Clearance 0.0625 inches (1.6 mm).

To avoid validity issues caused by precision loss, **ST_ReducePrecision** can reduce coordinate precision while ensuring that polygonal geometry remains valid.

2.3.0 ☒☒☒☒☒☒☒☒☒☒☒.


```
SELECT ST_MinimumClearance('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))');
st_minimumclearance
-----
0.00032
```

返回

[ST_MinimumClearanceLine](#), [ST_Crosses](#), [ST_Dimension](#), [ST_Intersects](#)

7.12.21 ST_MinimumClearanceLine

`ST_MinimumClearanceLine` — 返回 2 个点的 LineString，表示几何体的最小 clearance。

Synopsis

Geometry **ST_MinimumClearanceLine**(geometry g);

返回

Returns the two-point LineString spanning a geometry's minimum clearance. If the geometry does not have a minimum clearance, LINESTRING EMPTY is returned.

GEOS 版本

2.3.0 引入。GEOS 3.6.0 修复了 bug。

示例

```
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 geometry 类型，`ST_Surface`, `ST_MultiSurface` 返回 2 个点的 LineString。对于 geography 类型，返回 0 个点的 LineString。对于 geography 类型，使用 [ST_Length](#) 函数。对于 geometry 类型，使用 `ST_Length` 函数。

For geography types, the calculations are performed using the inverse geodesic problem, where perimeter units are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is

specified by the SRID, otherwise it is exclusive to WGS84. If `use_spheroid = false`, then calculations will approximate a sphere instead of a spheroid.

`ST_Perimeter2D` 函数, `ST_Perimeter` 函数.

✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1](#)

✔ This method implements the SQL/MM specification. SQL-MM 3: 8.1.3, 9.5.4

版本: 2.0.0 函数参考.

例:

计算一个多边形的周长。使用 EPSG:2249 坐标系。

```
SELECT ST_Perimeter(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416))', 2249));
st_perimeter
-----
122.630744000095
(1 row)
```

```
SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((763104.471273676 2949418.44119003,
763104.477769673 2949418.42538203,
763104.189609677 2949418.22343004,763104.471273676 2949418.44119003)),
((763104.471273676 2949418.44119003,763095.804579742 2949436.33850239,
763086.132105649 2949451.46730207,763078.452329651 2949462.11549407,
763075.354136904 2949466.17407812,763064.362142565 2949477.64291974,
763059.953961626 2949481.28983009,762994.637609571 2949532.04103014,
762990.568508415 2949535.06640477,762986.710889563 2949539.61421415,
763117.237897679 2949709.50493431,763235.236617789 2949617.95619822,
763287.718121842 2949562.20592617,763111.553321674 2949423.91664605,
763104.471273676 2949418.44119003)))', 2249));
st_perimeter
-----
845.227713366825
(1 row)
```

例:

计算一个多边形的周长。使用 WGS84 坐标系。

```
SELECT ST_Perimeter(geog) As per_meters, ST_Perimeter(geog)/0.3048 As per_ft
FROM ST_GeogFromText('POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326
42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.1776848522251
42.3902896512902))') As geog;

per_meters | per_ft
-----+-----
37.3790462565251 | 122.634666195949

-- MultiPolygon example --
SELECT ST_Perimeter(geog) As per_meters, ST_Perimeter(geog,false) As per_sphere_meters,
ST_Perimeter(geog)/0.3048 As per_ft
```

```
FROM ST_GeogFromText('MULTIPOLYGON((( -71.1044543107478 42.340674480411, -71.1044542869917 ↵
    42.3406744369506,
    -71.1044553562977 42.340673886454, -71.1044543107478 42.340674480411))),
    (( -71.1044543107478 42.340674480411, -71.1044860600303 42.3407237015564, -71.1045215770124 ↵
    42.3407653385914,
    -71.1045498002983 42.3407946553165, -71.1045611902745 42.3408058316308, -71.1046016507427 ↵
    42.340837442371,
    -71.104617893173 42.3408475056957, -71.1048586153981 42.3409875993595, -71.1048736143677 ↵
    42.3409959528211,
    -71.1048878050242 42.3410084812078, -71.1044020965803 42.3414730072048,
    -71.1039672113619 42.3412202916693, -71.1037740497748 42.3410666421308,
    -71.1044280218456 42.3406894151355, -71.1044543107478 42.340674480411))))') As geog;
```

per_meters	per_sphere_meters	per_ft
257.634283683311	257.412311446337	845.256836231335



ST GeogFromText, ST GeomFromText, ST Length

7.12.23 ST Perimeter2D

ST_Perimeter2D — Returns the 2D perimeter of a polygonal geometry. Alias for ST_Perimeter.

Synopsis

```
float ST_Perimeter2D(geometry geomA);
```

[illegible]

Note

```

XXXXXXXXXX ST_Perimeter XXXXXXXXXX. XXXXXXXXXX ST_Perimeter XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXXXXXXX.

```



ST Perimeter

7.12.24 ST 3DPerimeter

ST 3DPerimeter — .

Synopsis

```
float ST_3DPerimeter(geometry geomA);
```

函数

返回指定几何体的 3D 周长。2D 周长使用 `ST_Perimeter2D` 函数。



This function supports 3d and will not drop the z-index.



This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.1, 10.5

版本: 2.0.0 函数名: `ST_Perimeter3D` 返回类型: `double`。

函数

以下 SQL 查询演示了该函数的用法：

```
SELECT ST_3DPerimeter(geom), ST_Perimeter2d(geom), ST_Perimeter(geom) FROM
    (SELECT ST_GeomFromEWKT('SRID=2249;POLYGON((743238 2967416 2,743238 2967450 1,
    743265.625 2967416 1,743238 2967416 2))') As geom) As foo;
```

ST_3DPerimeter	st_perimeter2d	st_perimeter
105.465793597674	105.432997272188	105.432997272188

函数

`ST_GeomFromEWKT`, `ST_Perimeter`, `ST_Perimeter2D`

7.12.25 ST_ShortestLine

`ST_ShortestLine` — 返回两个几何体之间的最短 2D 线。

Synopsis

geometry **ST_ShortestLine**(geometry geom1, geometry geom2);
 geography **ST_ShortestLine**(geography geom1, geography geom2, boolean use_spheroid = true);

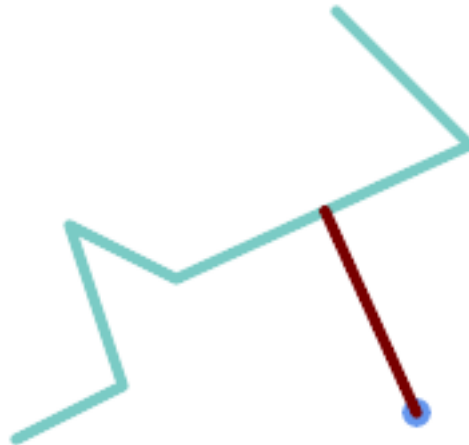
函数

返回两个几何体之间的 2D 最短线。返回的线从 `geom1` 开始并在 `geom2` 结束。如果 `geom1` 和 `geom2` 相交，则结果是具有开始和结束于交点的线。线的长度与 `ST_Distance` 返回的 `g1` 和 `g2` 相同。

增强: 3.4.0 - 支持 geography。

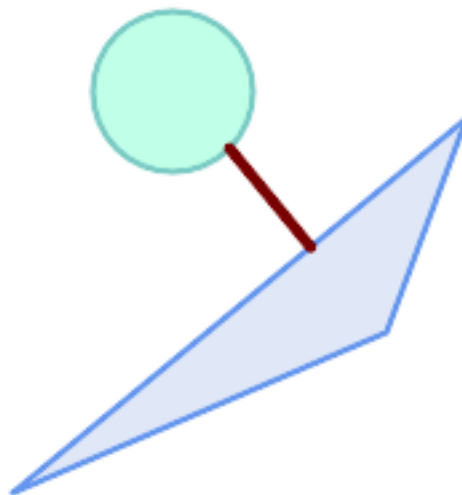
1.5.0 函数名: `ST_ShortestLine`。

图 10



Shortest line between Point and LineString

```
SELECT ST_AsText( ST_ShortestLine(
    'POINT (160 40)',
    'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)')
) As sline;
-----
LINESTRING(160 40,125.75342465753425 115.34246575342466)
```



Shortest line between Polygons

```
SELECT ST_AsText( ST_ShortestLine(
    'POLYGON ((190 150, 20 10, 160 70, 190 150))',
    ST_Buffer('POINT(80 160)', 30)
) ) AS llinevkt;
-----
LINESTRING(131.59149149528952 101.89887534906197,101.21320343559644 138.78679656440357)
```


☒☒☒☒☒☒☒☒☒☒ -- 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;
```

shl2d_line_pt	shl3d_line_pt
LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30)	LINESTRING(50 75.5074)

XXXXXXXXXXXXXXXX -- 3D, 2D XXXXXXXXXXXX

```
SELECT ST_AsEWKT(ST_3DShortestLine(poly, mline)) As shl3d,
       ST_AsEWKT(ST_ShortestLine(poly, mline)) As shl2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,
100 100 5, 175 150 5)))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125
100 1, 175 155 1),
       (1 10 2, 5 20 1)))') As mline ) As foo;
shl3d  ↵
-----|shl2d
LINESTRING(39.993580415989 54.1889925532825 5,40.4078575708294 53.6052383805529 ↵
5.03423778139177) | LINESTRING(20 40,20 40)
```



ST 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 ☒☒☒☒☒☒☒☒☒☒☒.

☐☐

```
-- Rely on implicit cast from geometry to box2d for the second parameter
SELECT ST_ClipByBox2D(geom, ST_MakeEnvelope(0,0,10,10)) FROM mytab;
```

☐☐

[ST_Intersection](#), [ST_MakeBox2D](#), [ST_MakeEnvelope](#)

7.13.2 ST_Difference

ST_Difference — Computes a geometry representing the part of geometry A that does not intersect geometry B.

Synopsis

geometry **ST_Difference**(geometry geomA, geometry geomB, float8 gridSize = -1);

☐☐

Returns a geometry representing the part of geometry A that does not intersect geometry B. This is equivalent to $A - \text{ST\_Intersection}(A,B)$. If A is completely contained in B then an empty atomic geometry of appropriate type is returned.



Note

This is the only overlay function where input order matters. **ST_Difference**(A, B) always returns a portion of A.

If the optional **gridSize** parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see [ST_ReducePrecision](#).

GEOS ☐☐☐☐☐

Enhanced: 3.1.0 accept a **gridSize** parameter.

Requires GEOS $\geq 3.9.0$ to use the **gridSize** parameter.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3](#)

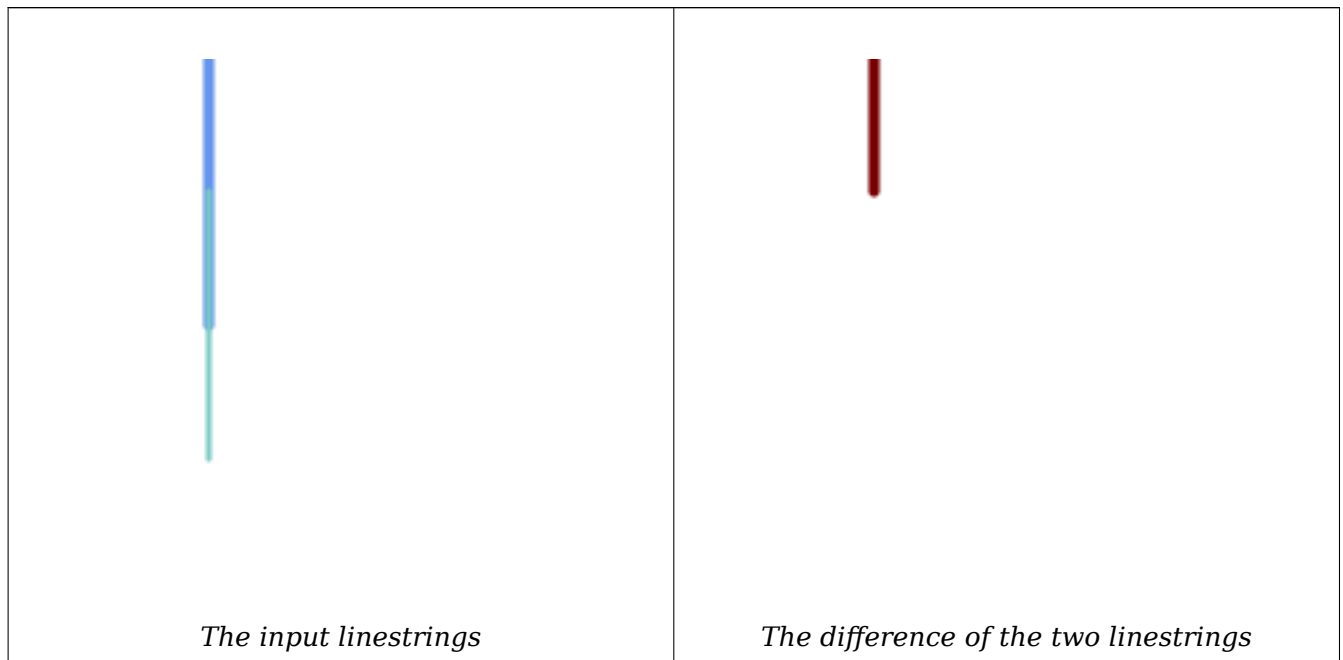


This method implements the SQL/MM specification. SQL-MM 3: 5.1.20



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

国国



The difference of 2D linestrings.

```
SELECT ST_AsText(
  ST_Difference(
    'LINESTRING(50 100, 50 200)::geometry',
    'LINESTRING(50 50, 50 150)::geometry'
  )
);
```

```
st_astext
-----
LINESTRING(50 150,50 200)
```

The difference of 3D points.

```
SELECT ST_AsEWKT( ST_Difference(
  'MULTIPOINT(-118.58 38.38 5,-118.60 38.329 6,-118.614 38.281 7)' :: geometry,
  'POINT(-118.614 38.281 5)' :: geometry
) );
```

```
st_asewkt
-----
MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)
```

国国

ST_SymDifference, ST_Intersection, ST_Union, ST_ReducePrecision

7.13.3 ST_Intersection

ST_Intersection — Computes a geometry representing the shared portion of geometries A and B.

Synopsis

```
geometry ST_Intersection( geometry geomA , geometry geomB , float8 gridSize = -1 );
geography ST_Intersection( geography geogA , geography geogB );
```

国国

Returns a geometry representing the point-set intersection of two geometries. In other words, that portion of geometry A and geometry B that is shared between the two geometries.

If the geometries have no points in common (i.e. are disjoint) then an empty atomic geometry of appropriate type is returned.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see [ST_ReducePrecision](#).

`ST_Intersection` in conjunction with [ST_Intersects](#) is useful for clipping geometries such as in bounding box, buffer, or region queries where you only require the portion of a geometry that is inside a country or region of interest.

Note



For geography this is a thin wrapper around the geometry implementation. It first determines the best SRID that fits the bounding box of the 2 geography objects (if geography objects are within one half zone UTM but not same UTM will pick one of those) (favoring UTM or Lambert Azimuthal Equal Area (LAEA) north/south pole, and falling back on mercator in worst case scenario) and then intersection in that best fit planar spatial ref and retransforms back to WGS84 geography.



Warning

This function will drop the M coordinate values if present.



Warning

If working with 3D geometries, you may want to use SFCGAL based [ST_3DIntersection](#) which does a proper 3D intersection for 3D geometries. Although this function works with Z-coordinate, it does an averaging of Z-Coordinate.

GEOS 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.

图

```
SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 )'::
  geometry));
  st_astext
-----
GEOMETRYCOLLECTION EMPTY

SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 )'::
  geometry));
  st_astext
-----
POINT(0 0)
```

Clip all lines (trails) by country. Here we assume country geom are POLYGON or MULTIPOLYGONS. NOTE: we are only keeping intersections that result in a LINESTRING or MULTILINESTRING because we don't care about trails that just share a point. The dump is needed to expand a geometry collection into individual single MULT* parts. The below is fairly generic and will work for polys, etc. by just changing the where clause.

```
select clipped.gid, clipped.f_name, clipped_geom
from (
    select trails.gid, trails.f_name,
        (ST_Dump(ST_Intersection(country.geom, trails.geom))).geom clipped_geom
    from country
        inner join trails on ST_Intersects(country.geom, trails.geom)
    ) as clipped
where ST_Dimension(clipped.clipped_geom) = 1;
```

For polys e.g. polygon landmarks, you can also use the sometimes faster hack that buffering anything by 0.0 except a polygon results in an empty geometry collection. (So a geometry collection containing polys, lines and points buffered by 0.0 would only leave the polygons and dissolve the collection shell.)

```
select poly.gid,
    ST_Multi(
        ST_Buffer(
            ST_Intersection(country.geom, poly.geom),
            0.0
        )
    ) clipped_geom
from country
    inner join poly on ST_Intersects(country.geom, poly.geom)
where not ST_IsEmpty(ST_Buffer(ST_Intersection(country.geom, poly.geom), 0.0));
```

Examples: 2.5Dish

Note this is not a true intersection, compare to the same example using [ST_3DIntersection](#).

```
select ST_AsText(ST_Intersection(linestring, polygon)) As wkt
from ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS
  linestring
CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

      st_astext
-----
LINESTRING Z (1 1 8,0.5 0.5 8,0 0 10)
```

☒☒

[ST_3DIntersection](#), [ST_Difference](#), [ST_Union](#), [ST_ClipByBox2D](#), [ST_Dimension](#), [ST_Dump](#), [ST_Force2D](#), [ST_SymDifference](#), [ST_Intersects](#), [ST_Multi](#), [ST_ReducePrecision](#)

7.13.4 ST_MemUnion

`ST_MemUnion` — Aggregate function which unions geometries in a memory-efficient but slower way

Synopsis

geometry **ST_MemUnion**(geometry set geomfield);

☒☒

An aggregate function that unions the input geometries, merging them to produce a result geometry with no overlaps. The output may be a single geometry, a MultiGeometry, or a Geometry Collection.



Note

Produces the same result as [ST_Union](#), but uses less memory and more processor time. This aggregate function works by unioning the geometries incrementally, as opposed to the `ST_Union` aggregate which first accumulates an array and then unions the contents using a fast algorithm.



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

☒☒

```
SELECT id,
       ST_MemUnion(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

☒☒

[ST_Union](#)

7.13.5 ST_Node

`ST_Node` — Nodes a collection of lines.

Synopsis

geometry **ST_Node**(geometry geom);



Returns a (Multi)LineString representing the fully noded version of a collection of linestrings. The noding preserves all of the input nodes, and introduces the least possible number of new nodes. The resulting linework is dissolved (duplicate lines are removed).

This is a good way to create fully-noded linework suitable for use as input to [ST_Polygonize](#).

[ST_UnaryUnion](#) can also be used to node and dissolve linework. It provides an option to specify a gridSize, which can provide simpler and more robust output. See also [ST_Union](#) for an aggregate variant.



This function supports 3d and will not drop the z-index.

GEOS 国国国国国

2.0.0 国国国国国国国国国国国国.

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;
output
-----
MULTILINESTRING Z ((0 0 0,5 5 4.5),(5 5 4.5,10 10 10,0 10 5,5 5 4.5),(5 5 4.5,10 0 3))
```

Noding two LineStrings which share common linework. Note that the result linework is dissolved.

```
SELECT ST_AsText(
    ST_Node('MULTILINESTRING ((2 5, 2 1, 7 1), (6 1, 4 1, 2 3, 2 5))::geometry')
) As output;
output
-----
MULTILINESTRING((2 5,2 3),(2 3,2 1,4 1),(4 1,2 3),(4 1,6 1),(6 1,7 1))
```



[ST_UnaryUnion](#), [ST_AsBinary](#)

7.13.6 ST_Split

ST_Split — Returns a collection of geometries created by splitting a geometry by another geometry.

Synopsis

geometry **ST_Split**(geometry input, geometry blade);



The function supports splitting a `LineString` by a `(Multi)Point`, `(Multi)LineString` or `(Multi)Polygon` boundary, or a `(Multi)Polygon` by a `LineString`. When a `(Multi)Polygon` is used as the blade, its linear components (the boundary) are used for splitting the input. The result geometry is always a collection.

This function is in a sense the opposite of `ST_Union`. Applying `ST_Union` to the returned collection should theoretically yield the original geometry (although due to numerical rounding this may not be exactly the case).



Note

If the the input and blade do not intersect due to numerical precision issues, the input may not be split as expected. To avoid this situation it may be necessary to snap the input to the blade first, using `ST_Snap` with a small tolerance.

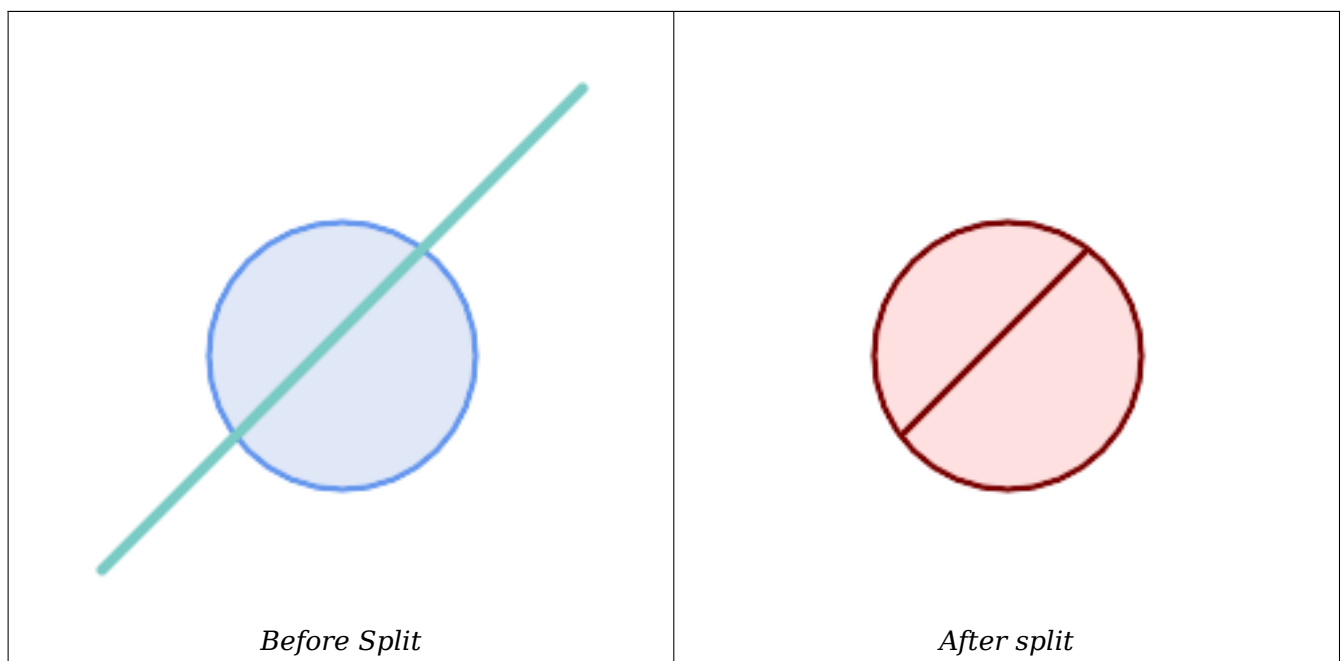
Availability: 2.0.0 requires GEOS

Enhanced: 2.2.0 support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced.

Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced.



Split a Polygon by a Line.

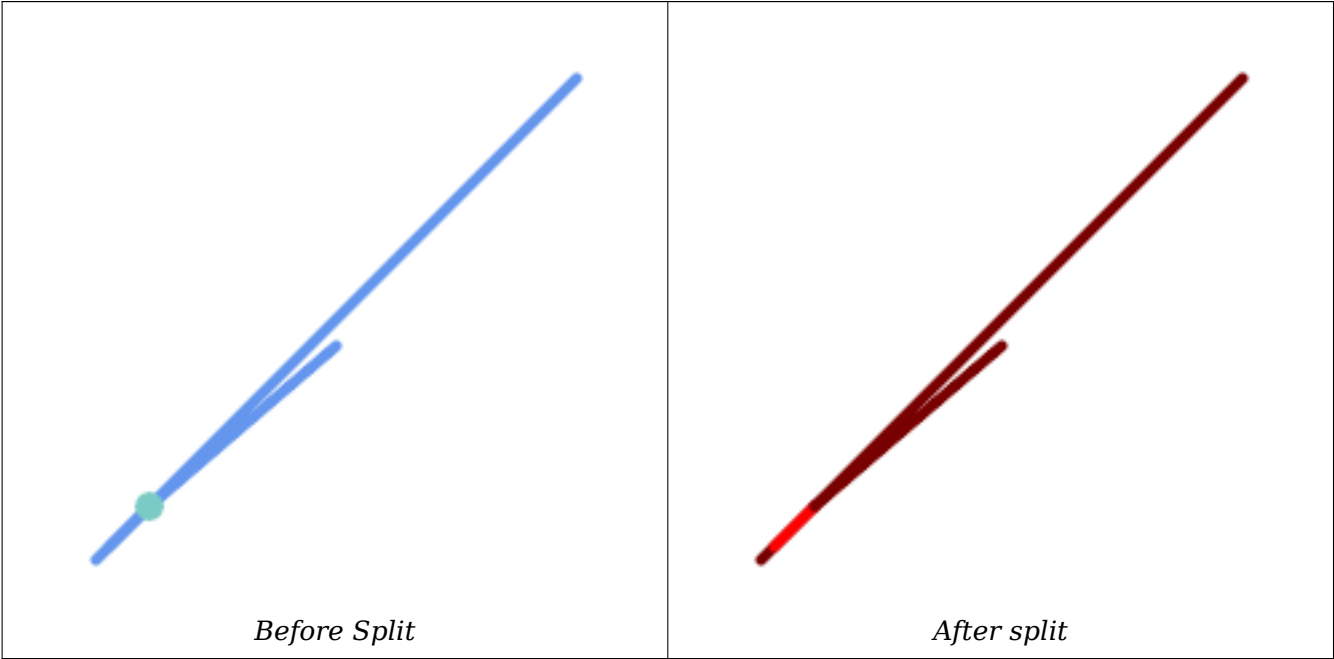


```
SELECT ST_AsText( ST_Split(
    ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50), -- circle
    ST_MakeLine(ST_Point(10, 10),ST_Point(190, 190)) -- line
));

-- result --
GEOMETRYCOLLECTION(
```

```
POLYGON((150 90,149.039264020162 80.2454838991936,146.193976625564 70.8658283817455,...),
POLYGON(...))
)
```

Split a MultiLineString by a Point, where the point lies exactly on both LineStrings elements.



```
SELECT ST_AsText(ST_Split(
  'MULTILINESTRING((10 10, 190 190), (15 15, 30 30, 100 90))',
  ST_Point(30,30))) As split;

split
-----
GEOMETRYCOLLECTION(
  LINESTRING(10 10,30 30),
  LINESTRING(30 30,190 190),
  LINESTRING(15 15,30 30),
  LINESTRING(30 30,100 90)
)
```

Split a LineString by a Point, where the point does not lie exactly on the line. Shows using **ST_Snap** to snap the line to the point to allow it to be split.

```
WITH data AS (SELECT
  'LINESTRING(0 0, 100 100)::geometry AS line,
  'POINT(51 50):: geometry AS point
)
SELECT ST_AsText( ST_Split( ST_Snap(line, point, 1), point)) AS snapped_split,
       ST_AsText( ST_Split(line, point)) AS not_snapped_not_split
FROM data;

snapped_split | not_snapped_not_split |
-----+-----
GEOMETRYCOLLECTION(LINESTRING(0 0,51 50),LINESTRING(51 50,100 100)) | GEOMETRYCOLLECTION(
  LINESTRING(0 0,100 100))
```

国国

[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);

国国

Returns a set of geometries that are the result of dividing geom into parts using rectilinear lines, with each part containing no more than max_vertices.

max_vertices must be 5 or more, as 5 points are needed to represent a closed box.

If the optional gridSize parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see [ST_ReducePrecision](#).

Point-in-polygon and other spatial operations are normally faster for indexed subdivided datasets. Since the bounding boxes for the parts usually cover a smaller area than the original geometry bbox, index queries produce fewer "hit" cases. The "hit" cases are faster because the spatial operations executed by the index recheck process fewer points.

Note



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 国国国国国

2.2.0 国国国国国国国国国国国国.

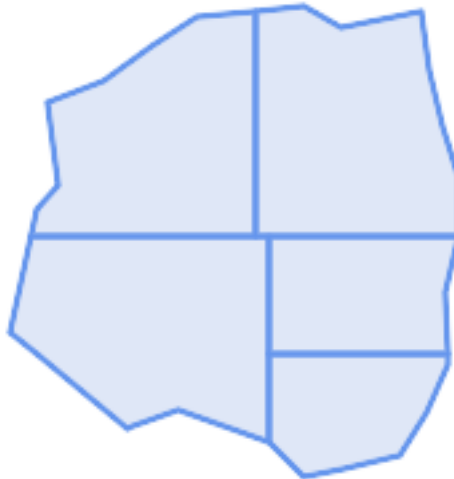
Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5.

Enhanced: 3.1.0 accept a gridSize parameter.

Requires GEOS >= 3.9.0 to use the gridSize parameter

图

Example: Subdivide a polygon into parts with no more than 10 vertices, and assign each part a unique id.

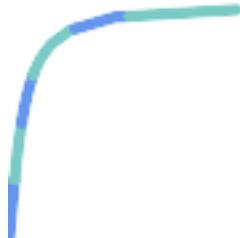


Subdivided to maximum 10 vertices

```
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
b''	b''-b''b''-b''b''-b''b''-b''b''-b''b''-+b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←	b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←
	'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←	'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←
	'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←	'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←
	'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←	'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←
	'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←	'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←
	'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←	'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←
	'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←	'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←
1	b'' b''	POLYGON((119 23,85 35,68 29,66 28,32 56,22 64,29.8260869565217 100,119 100,119 23)) ←
2	b'' b''	POLYGON((132 10,119 23,119 56,186 56,186 52,178 34,168 18,147 13,132 10))
3	b'' b''	POLYGON((119 56,119 100,190 100,185 79,186 56,119 56))
4	b'' b''	POLYGON((29.8260869565217 100,32 110,40 119,36 150,57 158,75 171,92 182,114 184,114 100,29.8260869565217 100)) ←
5	b'' b''	POLYGON((114 184,132 186,146 178,176 184,179 162,184 141,190 122,190 100,114 100,114 184)) ←

Example: Densify a long geography line using ST_Segmentize(geography, distance), and use ST_Subdivide to split the resulting line into sublines of 8 vertices. 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;
```

国国

[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);

国国

Returns a geometry representing the portions of geometries A and B that do not intersect. This is equivalent to `ST_Union(A,B) - ST_Intersection(A,B)`. It is called a symmetric difference because `ST_SymDifference(A,B) = ST_SymDifference(B,A)`.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see [ST_ReducePrecision](#).

GEOS 国国国国国

Enhanced: 3.1.0 accept a `gridSize` parameter.

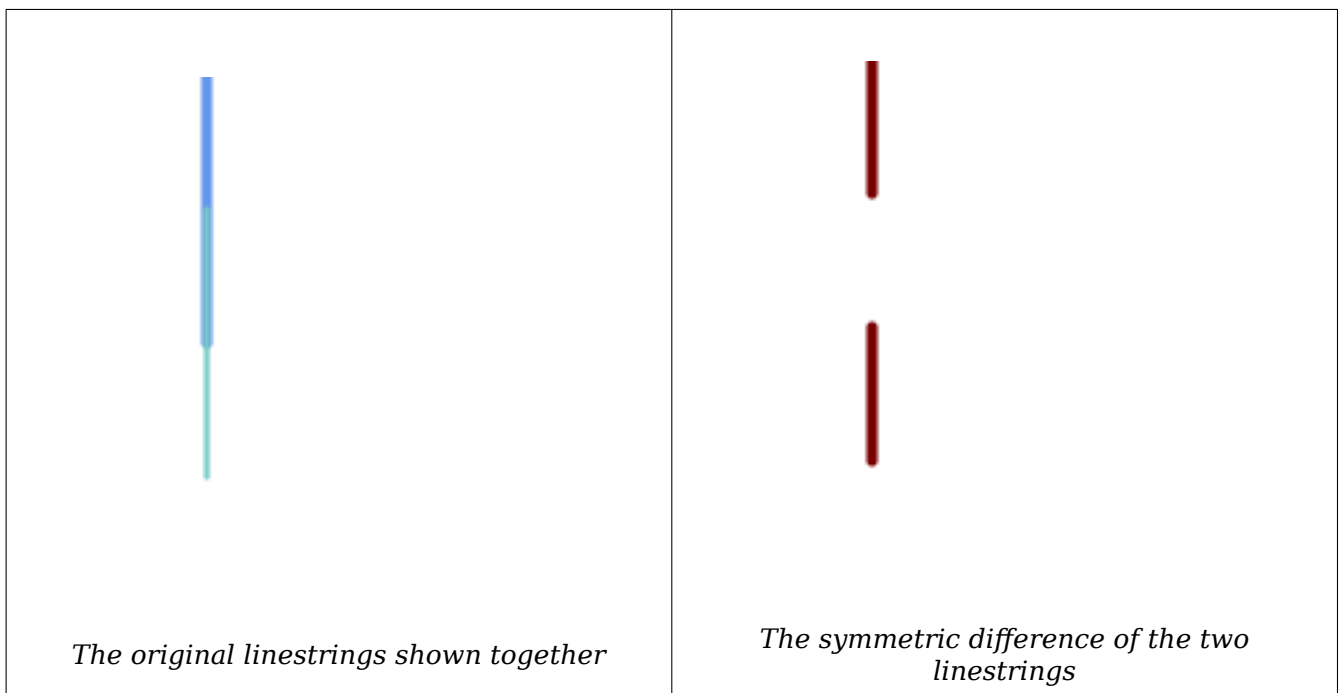
Requires GEOS >= 3.9.0 to use the `gridSize` parameter

✔ 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.

国国



```
--Safe for 2d - symmetric difference of 2 linestrings
SELECT ST_AsText(
  ST_SymDifference(
    ST_GeomFromText('LINESTRING(50 100, 50 200)'),
    ST_GeomFromText('LINESTRING(50 50, 50 150)')
  )
);

st_astext
-----
MULTILINESTRING((50 150,50 200),(50 50,50 100))
```

```
--When used in 3d doesn't quite do the right thing
SELECT ST_AsEWKT(ST_SymDifference(ST_GeomFromEWKT('LINESTRING(1 2 1, 1 4 2)'),
  ST_GeomFromEWKT('LINESTRING(1 1 3, 1 3 4)')));

st_astext
-----
MULTILINESTRING((1 3 2.75,1 4 2),(1 1 3,1 2 2.25))
```



ST_Difference, ST_Intersection, ST_Union, ST_ReducePrecision

7.13.9 ST_UnaryUnion

ST_UnaryUnion — Computes the union of the components of a single geometry.

Synopsis

geometry **ST_UnaryUnion**(geometry geom, float8 gridSize = -1);

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 grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see **ST_ReducePrecision**.



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Enhanced: 3.1.0 accept a `gridSize` parameter.

Requires GEOS >= 3.9.0 to use the `gridSize` parameter

2.0.0 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 grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see [ST_ReducePrecision](#).



Note

[ST_Collect](#) may sometimes be used in place of ST_Union, if the result is not required to be non-overlapping. ST_Collect is usually faster than ST_Union because it performs no processing on the collected geometries.

GEOS 国国国国国国

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.

国国

Aggregate example

```
SELECT id,
       ST_Union(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

Non-Aggregate example

```
select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(-2 3)' :: geometry))

st_astext
-----
MULTIPOINT(-2 3,1 2)

select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(1 2)' :: geometry))

st_astext
-----
POINT(1 2)
```

3D example - sort of supports 3D (and with mixed dimensions!)

```
select ST_AsEWKT(ST_Union(geom))
from (
    select 'POLYGON((-7 4.2,-7.1 4.2,-7.1 4.3, -7 4.2))'::geometry geom
    union all
    select 'POINT(5 5 5)'::geometry geom
    union all
    select 'POINT(-2 3 1)'::geometry geom
    union all
    select 'LINESTRING(5 5 5, 10 10 10)'::geometry geom
) as foo;

st_asewkt
-----
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 5,-7.1 4.2 5,-7.1 4.3 5,-7 4.2 5)));
```

3d example not mixing dimensions

```
select ST_AsEWKT(ST_Union(geom))
from (
    select 'POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2, -7 4.2 2))'::geometry geom
    union all
    select 'POINT(5 5 5)'::geometry geom
    union all
    select 'POINT(-2 3 1)'::geometry geom
    union all
    select 'LINESTRING(5 5 5, 10 10 10)'::geometry geom
) as foo;

st_asewkt
-----
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2,-7 4.2 2)));

--Examples using new Array construct
SELECT ST_Union(ARRAY(SELECT geom FROM sometable));

SELECT ST_AsText(ST_Union(ARRAY[ST_GeomFromText('LINESTRING(1 2, 3 4)'),
    ST_GeomFromText('LINESTRING(3 4, 4 5)')])) As wktunion;

--wktunion---
MULTILINESTRING((3 4,4 5),(1 2,3 4))
```

函数

[ST_Collect](#), [ST_UnaryUnion](#), [ST_MemUnion](#), [ST_Intersection](#), [ST_Difference](#), [ST_SymDifference](#), [ST_Reduce](#)

7.14 几何操作

7.14.1 ST_Buffer

ST_Buffer — Computes a geometry covering all points within a given distance from a geometry.

Synopsis

```
geometry ST_Buffer(geometry g1, float radius_of_buffer, text buffer_style_parameters = "");
geometry ST_Buffer(geometry g1, float radius_of_buffer, integer num_seg_quarter_circle);
geography ST_Buffer(geography g1, float radius_of_buffer, text buffer_style_parameters);
geography ST_Buffer(geography g1, float radius_of_buffer, integer num_seg_quarter_circle);
```

描述

Computes a POLYGON or MULTIPOLYGON that represents all points whose distance from a geometry/geography is less than or equal to a given distance. A negative distance shrinks the geometry rather than expanding it. A negative distance may shrink a polygon completely, in which case POLYGON EMPTY is returned. For points and lines negative distances always return empty results.

For geometry, the distance is specified in the units of the Spatial Reference System of the geometry. For geography, the distance is specified in meters.

The optional third parameter controls the buffer accuracy and style. The accuracy of circular arcs in the buffer is specified as the number of line segments used to approximate a quarter circle (default is 8). The buffer style can be specified by providing a list of blank-separated key=value pairs as follows:

- 'quad_segs=#' : number of line segments used to approximate a quarter circle (default is 8).
- 'endcap=round|flat|square' : endcap style (defaults to "round"). 'butt' is accepted as a synonym for 'flat'.
- 'join=round|mitre|bevel' : join style (defaults to "round"). 'miter' is accepted as a synonym for 'mitre'.
- 'mitre_limit=#.#' : mitre ratio limit (only affects mitered join style). 'miter_limit' is accepted as a synonym for 'mitre_limit'.
- 'side=both|left|right' : 'left' or 'right' performs a single-sided buffer on the geometry, with the buffered side relative to the direction of the line. This is only applicable to LINESTRING geometry and does not affect POINT or POLYGON geometries. By default end caps are square.

Note



For geography this is a thin wrapper around the geometry implementation. It determines a planar spatial reference system that best fits the bounding box of the geography object (trying UTM, Lambert Azimuthal Equal Area (LAEA) North/South pole, and finally Mercator). The buffer is computed in the planar space, and then transformed back to WGS84. This may not produce the desired behavior if the input object is much larger than a UTM zone or crosses the dateline

**Note**

Buffer can handle invalid inputs and the output is always a valid polygonal geometry. Buffering by distance 0 is sometimes used as a way of repairing invalid polygons. **ST_MakeValid** is more suitable for this process as it can handle multi-polygons.

**Note**

Buffering is sometimes used to perform a within-distance search. For this use case it is more efficient to use **ST_DWithin**.

**Note**

This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry.

Enhanced: 2.5.0 - ST_Buffer geometry support was enhanced to allow for side buffering specification `side=both|left|right`.

Availability: 1.5 - ST_Buffer was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added.

GEOS 国国国国国国

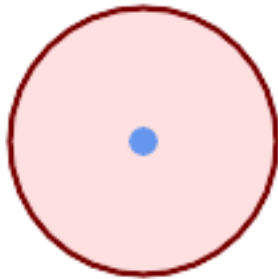


This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**, s2.1.1.3



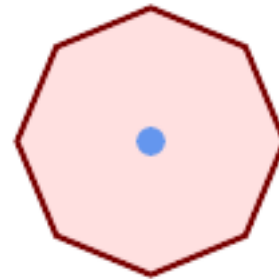
This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.30

国国



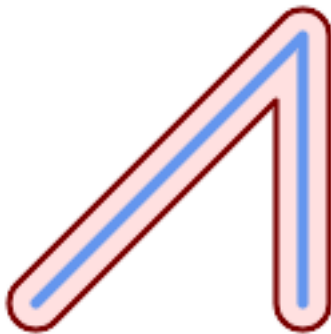
quad_segs=8 (图例)

```
SELECT ST_Buffer(
  ST_GeomFromText('POINT(100 90)'),
  50, 'quad_segs=8');
```



quad_segs=2 (图例)

```
SELECT ST_Buffer(
  ST_GeomFromText('POINT(100 90)'),
  50, 'quad_segs=2');
```



endcap=round join=round (图例)

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'endcap=round join=round');
```



endcap=square

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'endcap=square join=round');
```



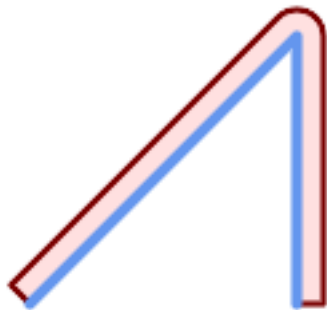
join=bevel

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'join=bevel');
```



join=mitre mitre_limit=5.0 (国国国国国国国国)

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'join=mitre mitre_limit=5.0');
```



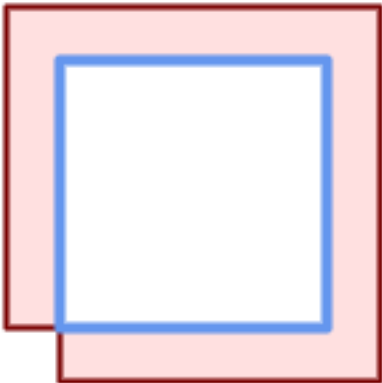
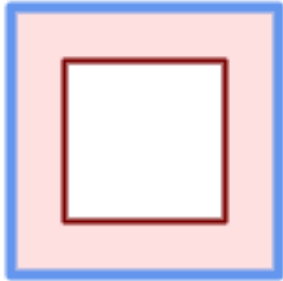
side=left

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'side=left');
```



side=right

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'side=right');
```

 <i>right-hand-winding, polygon boundary side=left</i> <pre>SELECT ST_Buffer(ST_ForceRHR(ST_Boundary(ST_GeomFromText('POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))')))) , 20, 'side=left');</pre>	 <i>right-hand-winding, polygon boundary side=right</i> <pre>SELECT ST_Buffer(ST_ForceRHR(ST_Boundary(ST_GeomFromText('POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))')))) , 20, 'side=right');</pre>

```
--A buffered point approximates a circle  
-- A buffered point forcing approximation of (see diagram)  
-- 2 points per quarter circle is poly with 8 sides (see diagram)  
SELECT ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50)) As promisingcircle_pcount,  
       ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 2)) As lamecircle_pcount;  
  
promisingcircle_pcount | lamecircle_pcount  
-----+-----  
33 | 9  
  
--A lighter but lamer circle  
-- only 2 points per quarter circle is an octagon  
--Below is a 100 meter octagon  
-- Note coordinates are in NAD 83 long lat which we transform  
to Mass state plane meter and then buffer to get measurements in meters;  
SELECT ST_AsText(ST_Buffer(  
  ST_Transform(  
    ST_SetSRID(ST_Point(-71.063526, 42.35785), 4269), 26986)  
    ,100,2)) As octagon;  
-----  
POLYGON((236057.59057465 900908.759918696,236028.301252769 900838.049240578,235  
957.59057465 900808.759918696,235886.879896532 900838.049240578,235857.59057465  
900908.759918696,235886.879896532 900979.470596815,235957.59057465 901008.759918  
696,236028.301252769 900979.470596815,236057.59057465 900908.759918696))
```


☒☒

[ST_Collect](#), [ST_DWithin](#), [ST_SetSRID](#), [ST_Transform](#), [ST_Union](#), [ST_MakeValid](#)

7.14.2 ST_BuildArea

ST_BuildArea — Creates a polygonal geometry formed by the linework of a geometry.

Synopsis

geometry **ST_BuildArea**(geometry geom);

☒☒

Creates an areal geometry formed by the constituent linework of the input geometry. The input can be a LineString, MultiLineString, Polygon, MultiPolygon or a GeometryCollection. The result is a Polygon or MultiPolygon, depending on input. If the input linework does not form polygons, NULL is returned.

Unlike [ST_MakePolygon](#), this function accepts rings formed by multiple lines, and can form any number of polygons.

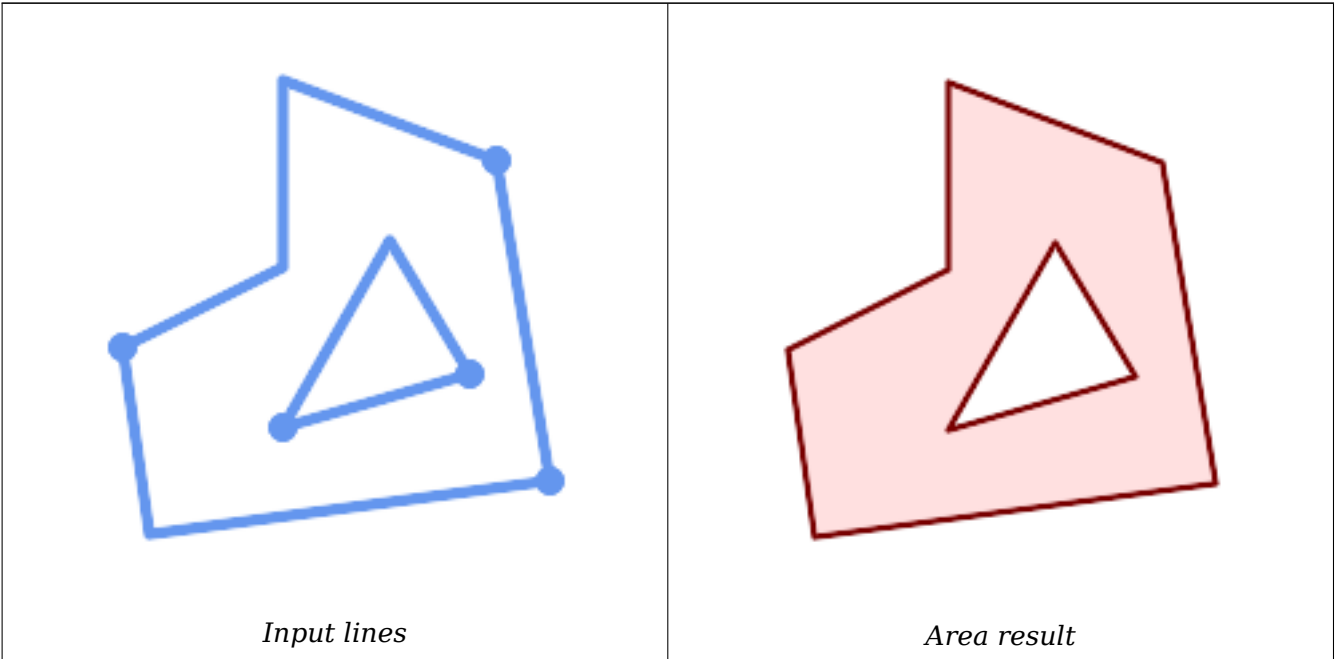
This function converts inner rings into holes. To turn inner rings into polygons as well, use [ST_Polygonize](#).



Note
Input linework must be correctly noded for this function to work properly. [ST_Node](#) can be used to node lines.
If the input linework crosses, this function will produce invalid polygons. [ST_MakeValid](#) can be used to ensure the output is valid.

1.1.0 简体中文

☒☒



```

WITH data(geom) AS (VALUES
  ('LINESTRING (180 40, 30 20, 20 90)::geometry')
, ('LINESTRING (180 40, 160 160)::geometry')
, ('LINESTRING (160 160, 80 190, 80 120, 20 90)::geometry')
, ('LINESTRING (80 60, 120 130, 150 80)::geometry')
, ('LINESTRING (80 60, 150 80)::geometry')
)
SELECT ST_AsText( ST_BuildArea( ST_Collect( geom )))
FROM data;
-----
POLYGON((180 40,30 20,20 90,80 120,80 190,160 160,180 40),(150 80,120 130,80 60,150 80))

```



Create a donut from two circular polygons

```

SELECT ST_BuildArea(ST_Collect(inring,outring))
FROM (SELECT
  ST_Buffer('POINT(100 90)', 25) As inring,
  ST_Buffer('POINT(100 90)', 50) As outring) As t;

```

☒☒

[ST_Collect](#), [ST_MakePolygon](#), [ST_MakeValid](#), [ST_Node](#), [ST_Polygonize](#), [ST_BdPolyFromText](#), [ST_BdMPolyFr](#)
(wrappers to this function with standard OGC interface)

7.14.3 ST_Centroid

ST_Centroid — ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

Synopsis

```

geometry ST_Centroid(geometry g1);
geography ST_Centroid(geography g1, boolean use_spheroid = true);

```

ST_Centroid

Computes a point which is the geometric center of mass of a geometry. For [MULTI]POINTS, the centroid is the arithmetic mean of the input coordinates. For [MULTI]LINESTRINGS, the centroid is computed using the weighted length of each line segment. For [MULTI]POLYGONS, the centroid is computed in terms of area. If an empty geometry is supplied, an empty GEOMETRYCOLLECTION is returned. If NULL is supplied, NULL is returned. If CIRCULARSTRING or COMPOUNDCURVE are supplied, they are converted to linestring with CurveToLine first, then same than for LINESTRING

For mixed-dimension input, the result is equal to the centroid of the component Geometries of highest dimension (since the lower-dimension geometries contribute zero "weight" to the centroid).

Note that for polygonal geometries the centroid does not necessarily lie in the interior of the polygon. For example, see the diagram below of the centroid of a C-shaped polygon. To construct a point guaranteed to lie in the interior of a polygon use [ST_PointOnSurface](#).

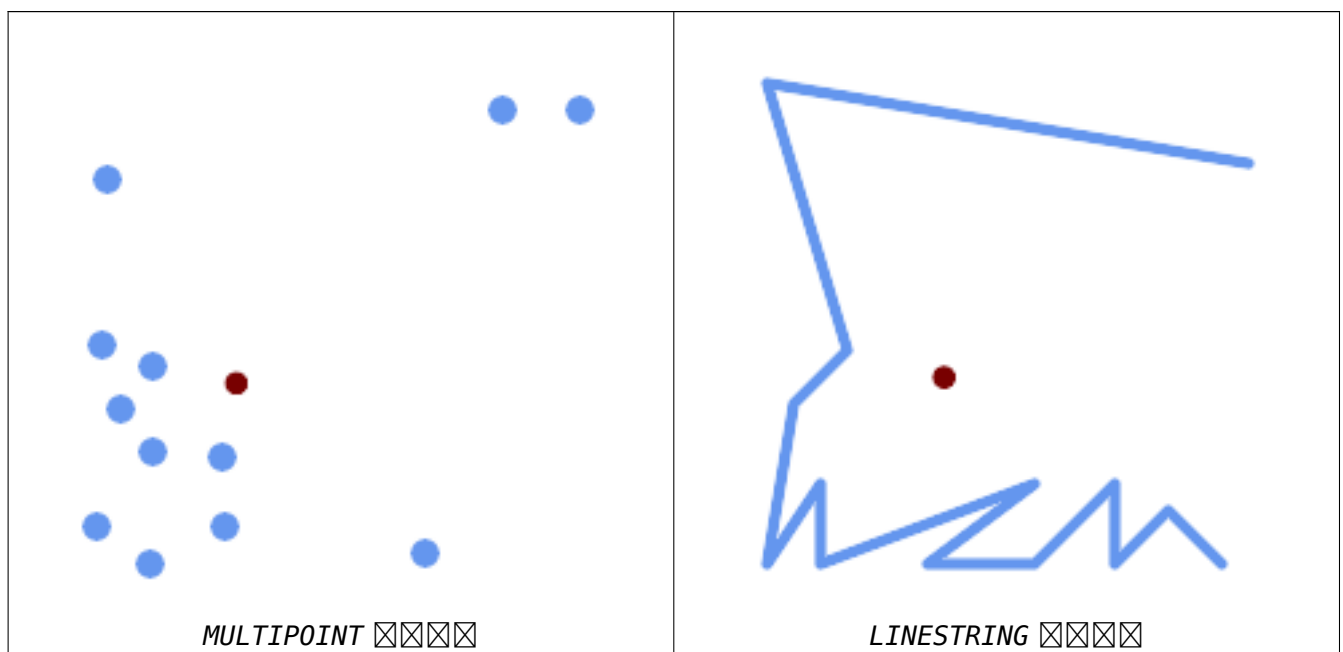
New in 2.3.0 : supports CIRCULARSTRING and COMPOUNDCURVE (using CurveToLine)

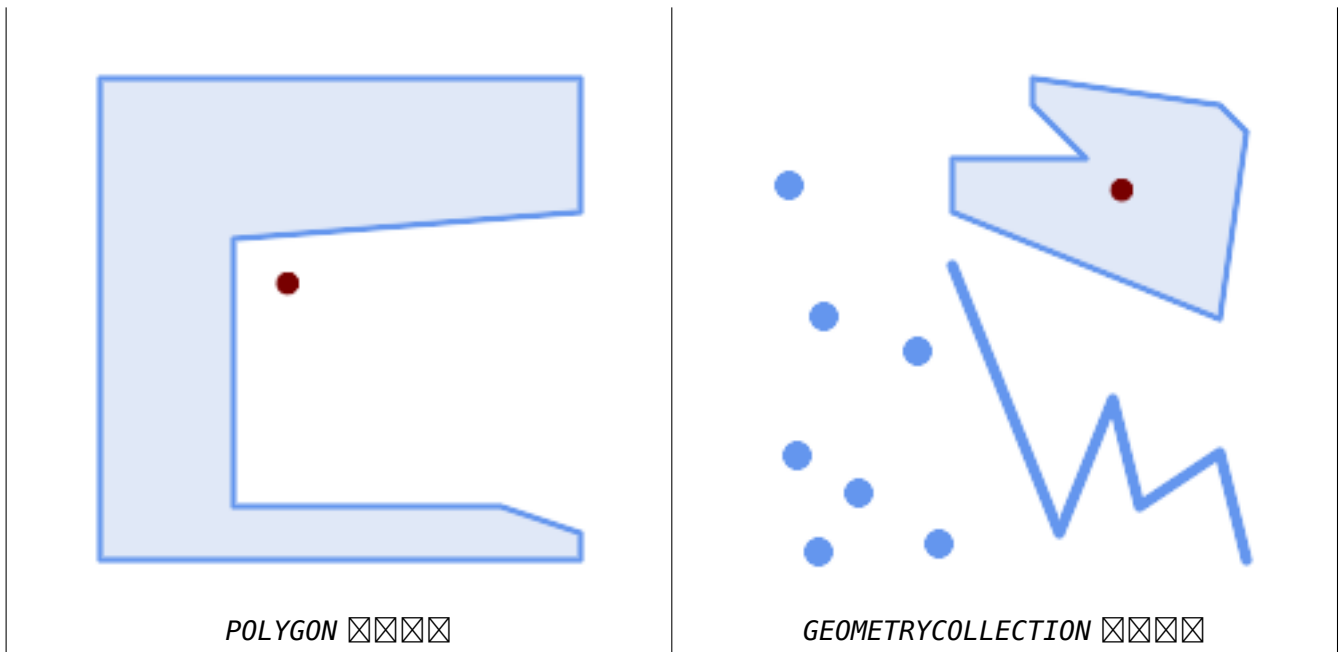
Availability: 2.4.0 support for geography was introduced.

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5

Examples

In the following illustrations the red dot is the centroid of the source geometry.





```
SELECT ST_AsText(ST_Centroid('MULTIPOINT ( -1 0, -1 2, -1 3, -1 4, -1 7, 0 1, 0 3, 1 1, 2 0, 6 0, 7 8, 9 8, 10 6 )'));
```

```
-----
POINT(2.30769230769231 3.30769230769231)
(1 row)
```

```
SELECT ST_AsText(ST_centroid(g))
FROM   ST_GeomFromText('CIRCULARSTRING(0 2, -1 1,0 0, 0.5 0, 1 0, 2 1, 1 2, 0.5 2, 0 2)') AS g ;
```

```
-----
POINT(0.5 1)
```

```
SELECT ST_AsText(ST_centroid(g))
FROM   ST_GeomFromText('COMPOUNDCURVE(CIRCULARSTRING(0 2, -1 1,0 0),(0 0, 0.5 0, 1 0), CIRCULARSTRING( 1 0, 2 1, 1 2),(1 2, 0.5 2, 0 2))' ) AS g;
```

```
-----
POINT(0.5 1)
```

图元

[ST_PointOnSurface](#), [ST_GeometricMedian](#)

7.14.4 ST_ChaikinSmoothing

ST_ChaikinSmoothing — Returns a smoothed version of a geometry, using the Chaikin algorithm

Synopsis

geometry **ST_ChaikinSmoothing**(geometry geom, integer nIterations = 1, boolean preserveEndpoints = false);



Smooths a linear or polygonal geometry using **Chaikin's algorithm**. The degree of smoothing is controlled by the `nIterations` parameter. On each iteration, each interior vertex is replaced by two vertices located at 1/4 of the length of the line segments before and after the vertex. A reasonable degree of smoothing is provided by 3 iterations; the maximum is limited to 5.

If `preserveEndPoints` is true, the endpoints of Polygon rings are not smoothed. The endpoints of LineStrings are always preserved.



Note

The number of vertices doubles with each iteration, so the result geometry may have many more points than the input. To reduce the number of points use a simplification function on the result (see [ST_Simplify](#), [ST_SimplifyPreserveTopology](#) and [ST_SimplifyVW](#)).

The result has interpolated values for the Z and M dimensions when present.



This function supports 3d and will not drop the z-index.

Availability: 2.5.0



Smoothing a triangle:

```
SELECT ST_AsText(ST_ChaikinSmoothing(geom)) smoothed
FROM (SELECT 'POLYGON((0 0, 8 8, 0 16, 0 0))'::geometry geom) AS foo;
```

smoothed

b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' ' ←

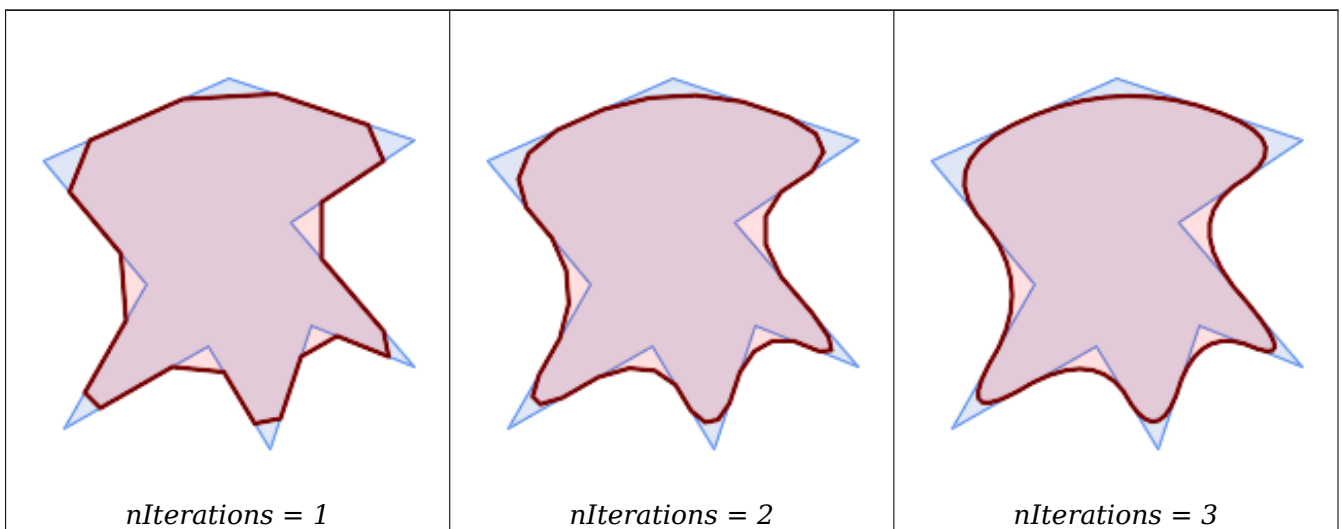
b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' ' ←

'-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' ' ←

'-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' ' ←

POLYGON((2 2,6 6,6 10,2 14,0 12,0 4,2 2))

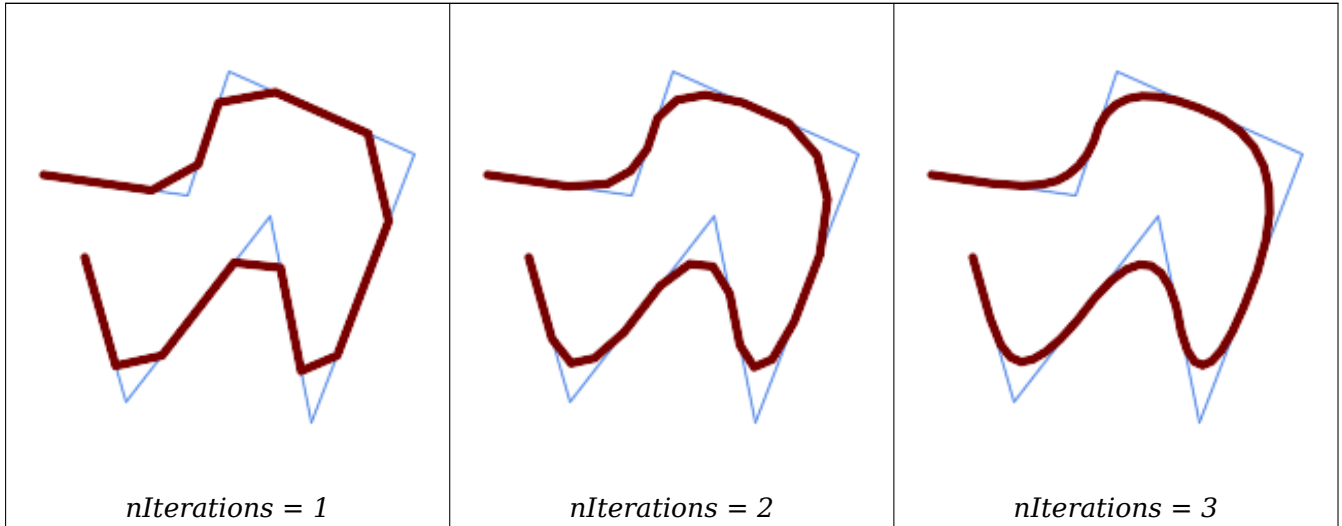
Smoothing a Polygon using 1, 2 and 3 iterations:



```
SELECT ST_ChaikinSmoothing(
```

```
'POLYGON ((20 20, 60 90, 10 150, 100 190, 190 160, 130 120, 190 50, 140 70, 120 10, 90 60, 20 20))',
generate_series(1, 3) );
```

Smoothing a LineString using 1, 2 and 3 iterations:



```
SELECT ST_ChaikinSmoothing(
  'LINESTRING (10 140, 80 130, 100 190, 190 150, 140 20, 120 120, 50 30, 30 100)'
  ,
  generate_series(1, 3) );
```

☒☒

ST_Simplify, **ST_SimplifyPreserveTopology**, **ST_SimplifyVW**

7.14.5 ST_ConcaveHull

ST_ConcaveHull — Computes a possibly concave geometry that contains all input geometry vertices

Synopsis

geometry **ST_ConcaveHull**(geometry param_geom, float param_pctconvex, boolean param_allow_holes = false);

☒☒

A concave hull is a (usually) concave geometry which contains the input, and whose vertices are a subset of the input vertices. In the general case the concave hull is a Polygon. The concave hull of two or more collinear points is a two-point LineString. The concave hull of one or more identical points is a Point. The polygon will not contain holes unless the optional `param_allow_holes` argument is specified as true.

One can think of a concave hull as “shrink-wrapping” a set of points. This is different to the **convex hull**, which is more like wrapping a rubber band around the points. A concave hull generally has a smaller area and represents a more natural boundary for the input points.

The `param_pctconvex` controls the concaveness of the computed hull. A value of 1 produces the convex hull. Values between 1 and 0 produce hulls of increasing concaveness. A value of 0 produces a hull with maximum concaveness (but still a single polygon). Choosing a suitable value depends on the nature of the input data, but often values between 0.3 and 0.1 produce reasonable results.



Note

Technically, the `param_pctconvex` determines a length as a fraction of the difference between the longest and shortest edges in the Delaunay Triangulation of the input points. Edges longer than this length are "eroded" from the triangulation. The triangles remaining form the concave hull.

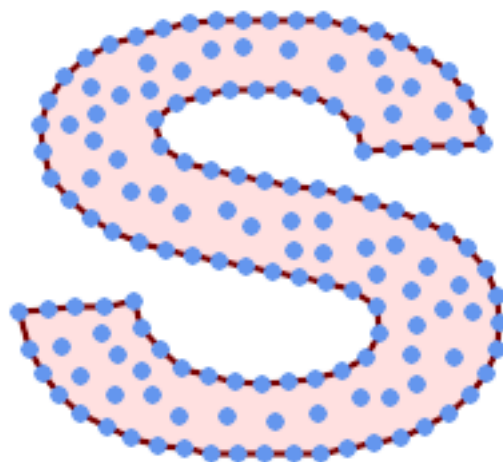
For point and linear inputs, the hull will enclose all the points of the inputs. For polygonal inputs, the hull will enclose all the points of the input *and also* all the areas covered by the input. If you want a point-wise hull of a polygonal input, convert it to points first using [ST_Points](#).

This is not an aggregate function. To compute the concave hull of a set of geometries use [ST_Collect](#) (e.g. `ST_ConcaveHull( ST_Collect( geom ), 0.80)`).

2.0.0 国国国国国国国国国国国国.

Enhanced: 3.3.0, GEOS native implementation enabled for GEOS 3.11+

国国



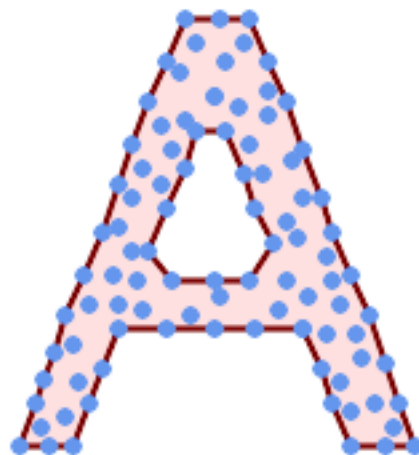
Concave Hull of a MultiPoint

```
SELECT ST_AsText( ST_ConcaveHull(
  'MULTIPOINT ((10 72), (53 76), (56 66), (63 58), (71 51), (81 48), (91 46), (101 45), (111 46), (121 47), (131 50), (140 55), (145 64), (144 74), (135 80), (125 83), (115 85), (105 87), (95 89), (85 91), (75 93), (65 95), (55 98), (45 102), (37 107), (29 114), (22 122), (19 132), (18 142), (21 151), (27 160), (35 167), (44 172), (54 175), (64 178), (74 180), (84 181), (94 181), (104 181), (114 181), (124 181), (134 179), (144 177), (153 173), (162 168), (171 162), (177 154), (182 145), (184 135), (139 132), (136 142), (128 149), (119 153), (109 155), (99 155), (89 155), (79 153), (69 150), (61 144), (63 134), (72 128), (82 125), (92 123), (102 121), (112 119), (122 118), (132 116), (142 113), (151 110), (161 106), (170 102), (178 96), (185 88), (189 78), (190 68), (189 58), (185 49), (179 41), (171 34), (162 29), (153 25), (143 23), (133 21), (123 19), (113 19), (102 19), (92 19), (82 19), (72 21), (62 22), (52 25), (43 29), (33 34), (25 41))
```

```

    , (19 49), (14 58), (21 73), (31 74), (42 74), (173 134), (161 134), (150 133), ←
    (97 104), (52 117), (157 156), (94 171), (112 106), (169 73), (58 165), (149 40) ←
    , (70 33), (147 157), (48 153), (140 96), (47 129), (173 55), (144 86), (159 67) ←
    , (150 146), (38 136), (111 170), (124 94), (26 59), (60 41), (71 162), (41 64), ←
    (88 110), (122 34), (151 97), (157 56), (39 146), (88 33), (159 45), (47 56), ←
    (138 40), (129 165), (33 48), (106 31), (169 147), (37 122), (71 109), (163 89), ←
    (37 156), (82 170), (180 72), (29 142), (46 41), (59 155), (124 106), (157 80), ←
    (175 82), (56 50), (62 116), (113 95), (144 167))',
    0.1 ) );
---st_astext--
POLYGON ((18 142, 21 151, 27 160, 35 167, 44 172, 54 175, 64 178, 74 180, 84 181, 94 181, ←
    104 181, 114 181, 124 181, 134 179, 144 177, 153 173, 162 168, 171 162, 177 154, 182 ←
    145, 184 135, 173 134, 161 134, 150 133, 139 132, 136 142, 128 149, 119 153, 109 155, 99 ←
    155, 89 155, 79 153, 69 150, 61 144, 63 134, 72 128, 82 125, 92 123, 102 121, 112 119, ←
    122 118, 132 116, 142 113, 151 110, 161 106, 170 102, 178 96, 185 88, 189 78, 190 68, ←
    189 58, 185 49, 179 41, 171 34, 162 29, 153 25, 143 23, 133 21, 123 19, 113 19, 102 19, ←
    92 19, 82 19, 72 21, 62 22, 52 25, 43 29, 33 34, 25 41, 19 49, 14 58, 10 72, 21 73, 31 ←
    74, 42 74, 53 76, 56 66, 63 58, 71 51, 81 48, 91 46, 101 45, 111 46, 121 47, 131 50, 140 ←
    55, 145 64, 144 74, 135 80, 125 83, 115 85, 105 87, 95 89, 85 91, 75 93, 65 95, 55 98, ←
    45 102, 37 107, 29 114, 22 122, 19 132, 18 142))

```



Concave Hull of a MultiPoint, allowing holes

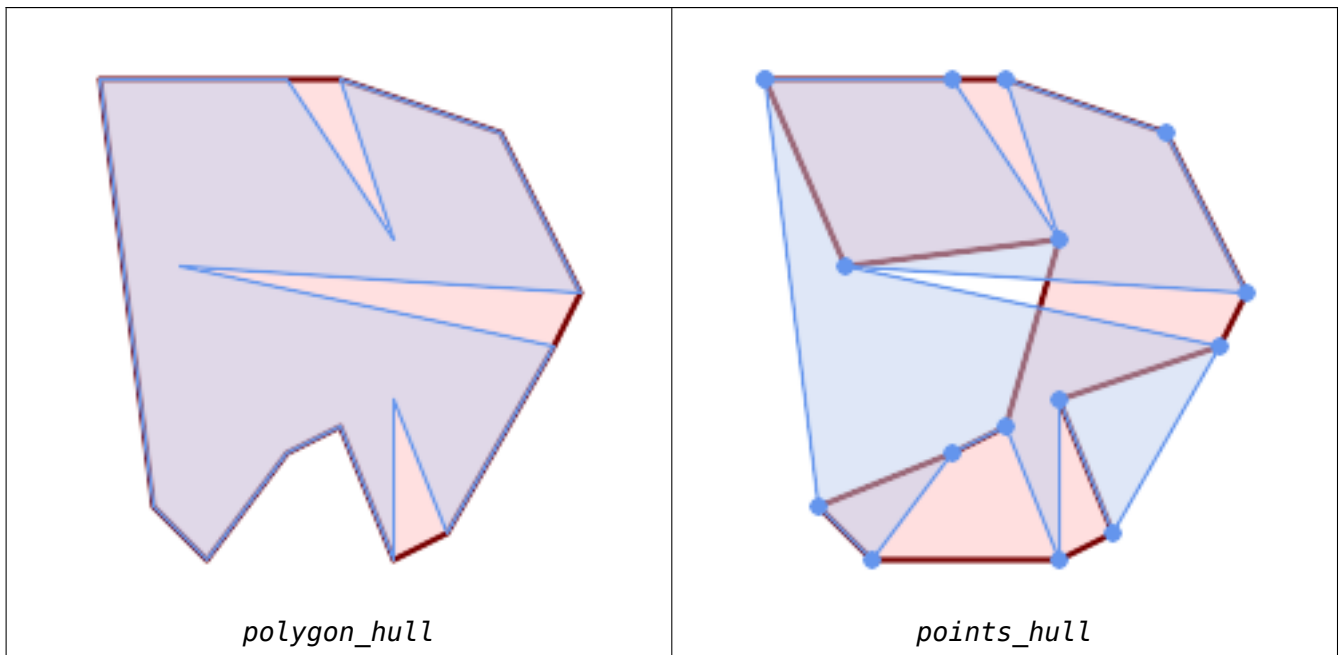
```

SELECT ST_AsText( ST_ConcaveHull(
    'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 36), (46 ←
    20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 100), (63 118), ←
    (68 133), (74 149), (81 164), (88 180), (101 180), (112 180), (119 164), (126 ←
    149), (132 131), (139 113), (143 100), (150 84), (157 69), (163 51), (168 36), ←
    (174 20), (163 20), (150 20), (143 36), (139 49), (132 64), (99 151), (92 138), ←
    (88 124), (81 109), (74 93), (70 82), (83 82), (99 82), (112 82), (126 82), (121 ←
    96), (114 109), (110 122), (103 138), (99 151), (34 27), (43 31), (48 44), (46 ←
    58), (52 73), (63 73), (61 84), (72 71), (90 69), (101 76), (123 71), (141 62), ←
    (166 27), (150 33), (159 36), (146 44), (154 53), (152 62), (146 73), (134 76), ←
    (143 82), (141 91), (130 98), (126 104), (132 113), (128 127), (117 122), (112 ←
    133), (119 144), (108 147), (119 153), (110 171), (103 164), (92 171), (86 160), ←
    (88 142), (79 140), (72 124), (83 131), (79 118), (68 113), (63 102), (68 93), ←
    (35 45)))',
    0.15, true ) );
---st_astext--
POLYGON ((43 69, 50 84, 57 100, 63 118, 68 133, 74 149, 81 164, 88 180, 101 180, 112 180, ←
    119 164, 126 149, 132 131, 139 113, 143 100, 150 84, 157 69, 163 51, 168 36, 174 20, 163 ←

```



```
20, 150 20, 143 36, 139 49, 132 64, 114 64, 99 64, 81 64, 63 64, 57 49, 52 36, 46 20, ↵
37 20, 26 20, 32 36, 35 45, 39 55, 43 69), (88 124, 81 109, 74 93, 83 82, 99 82, 112 82, ↵
121 96, 114 109, 110 122, 103 138, 92 138, 88 124))
```



Comparing a concave hull of a Polygon to the concave hull of the constituent points. The hull respects the boundary of the polygon, whereas the points-based hull does not.

```
WITH data(geom) AS (VALUES
  ('POLYGON ((10 90, 39 85, 61 79, 50 90, 80 80, 95 55, 25 60, 90 45, 70 16, 63 38, 60 10, ↵
    50 30, 43 27, 30 10, 20 20, 10 90))'::geometry)
)
SELECT ST_ConcaveHull( geom, 0.1) AS polygon_hull,
       ST_ConcaveHull( ST_Points(geom), 0.1) AS points_hull
FROM data;
```

Using with `ST_Collect` to compute the concave hull of a geometry set.

```
-- Compute estimate of infected area based on point observations
SELECT disease_type,
       ST_ConcaveHull( ST_Collect(obs_pnt), 0.3 ) AS geom
FROM disease_obs
GROUP BY disease_type;
```

☒☒

[ST_ConvexHull](#), [ST_Collect](#), [ST_AlphaShape](#), [ST_OptimalAlphaShape](#)

7.14.6 ST_ConvexHull

`ST_ConvexHull` — Computes the convex hull of a geometry.

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

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.

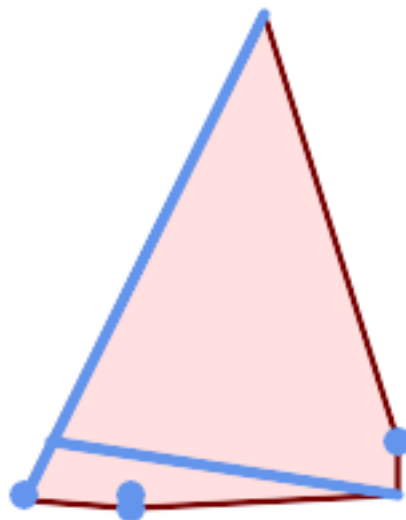
In the general case the convex hull is a Polygon. The convex hull of two or more collinear points is a two-point LineString. The convex hull of one or more identical points is a Point.

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.

```
--Get estimate of infected area based on point observations
SELECT d.disease_type,
       ST_ConvexHull(ST_Collect(d.geom)) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```

☐☐

[ST_Collect](#), [ST_ConcaveHull](#), [ST_MinimumBoundingCircle](#)

7.14.7 ST_DelaunayTriangles

`ST_DelaunayTriangles` — Returns the Delaunay triangulation of the vertices of a geometry.

Synopsis

geometry **ST_DelaunayTriangles**(geometry g1, float tolerance = 0.0, int4 flags = 0);

☐☐

Computes the [Delaunay triangulation](#) of the vertices of the input geometry. The optional tolerance can be used to snap nearby input vertices together, which improves robustness in some situations. The result geometry is bounded by the convex hull of the input vertices. The result geometry representation is determined by the flags code:

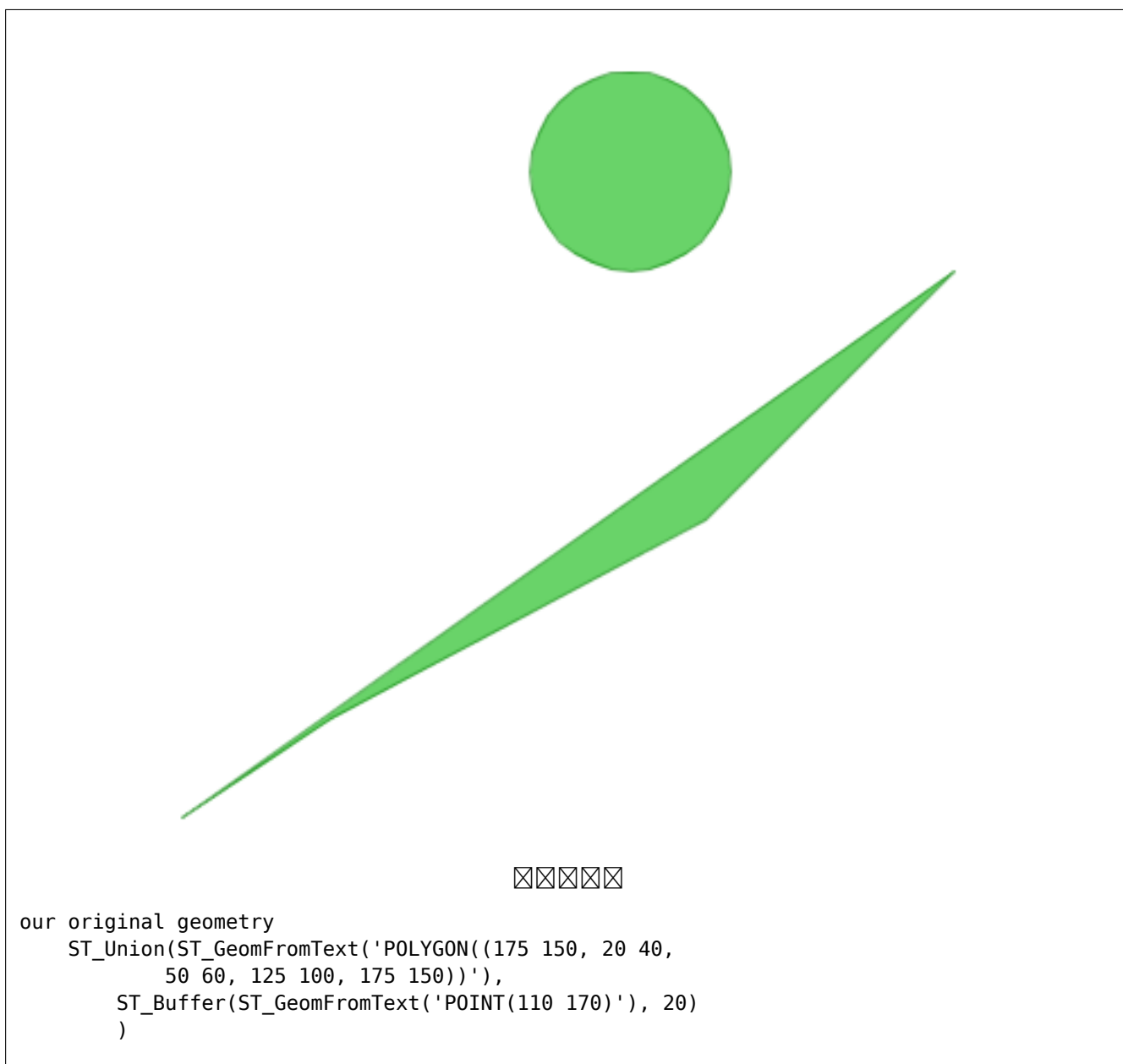
- 0 - a GEOMETRYCOLLECTION of triangular POLYGONS (default)
- 1 - a MULTILINESTRING of the edges of the triangulation
- 2 - A TIN of the triangulation

GEOS ☐☐☐☐☐

2.1.0 ☐☐☐☐☐☐☐☐☐☐.

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☐☐



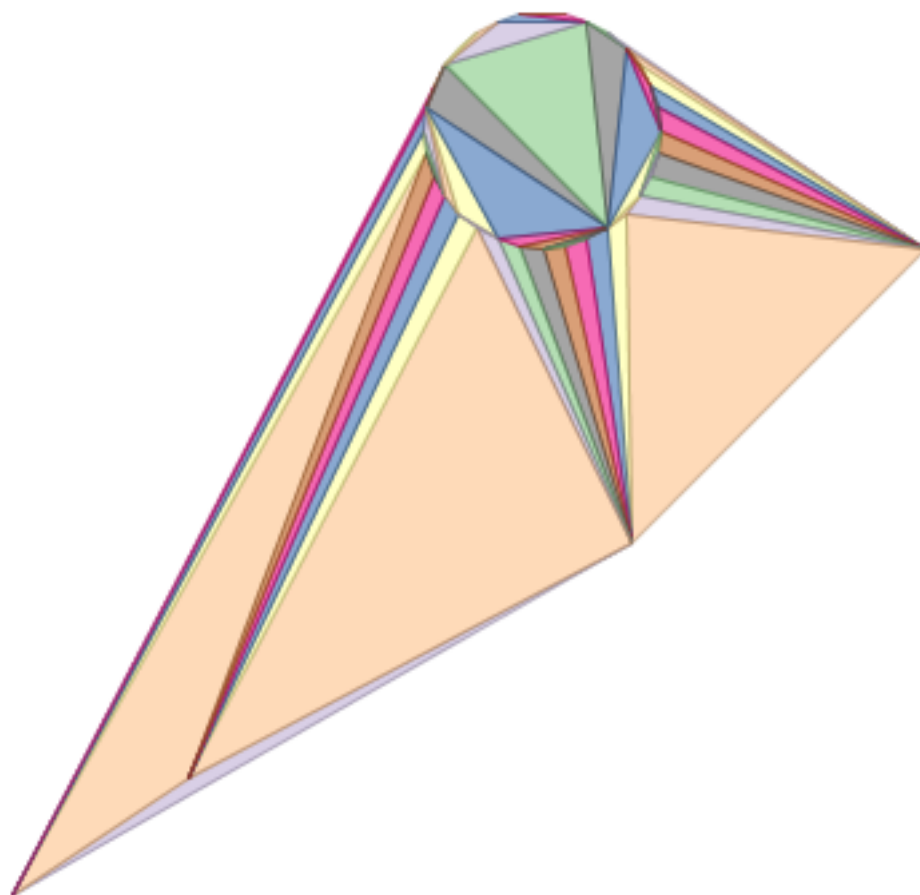
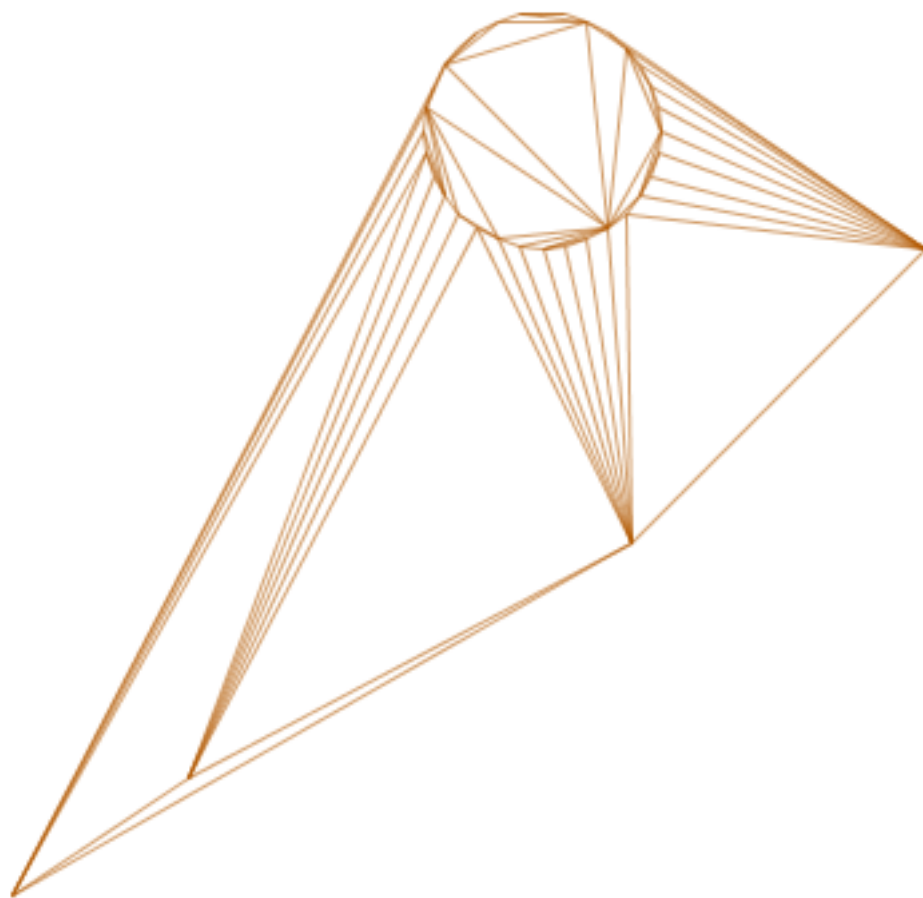


图 10-10 *ST_DelaunayTriangles*: 对多边形进行三角剖分

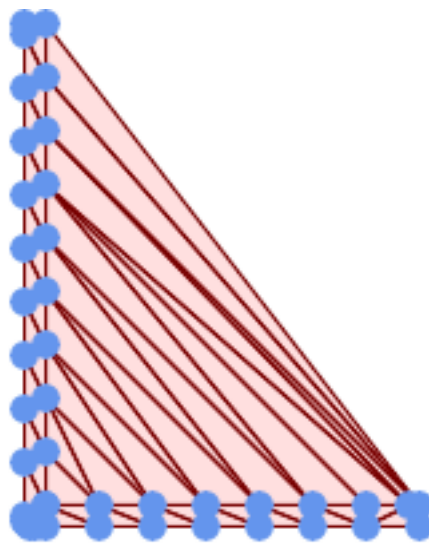
geometries overlaid multilinestring triangles

```
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;
```



-- 简体中文版

```
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;
```



```
-- 设置 55 度 45 度 设置
```

this produces a table of 42 points that form an L shape

```
SELECT (ST_DumpPoints(ST_GeomFromText(
'MULTIPOINT(14 14,34 14,54 14,74 14,94 14,114 14,134 14,
150 14,154 14,154 6,134 6,114 6,94 6,74 6,54 6,34 6,
14 6,10 6,8 6,7 7,6 8,6 10,6 30,6 50,6 70,6 90,6 110,6 130,
6 150,6 170,6 190,6 194,14 194,14 174,14 154,14 134,14 114,
14 94,14 74,14 54,14 34,14 14)'))).geom
    INTO TABLE l_shape;
```

output as individual polygon triangles

```
SELECT ST_AsText((ST_Dump(geom)).geom) As wkt
FROM ( SELECT ST_DelaunayTriangles(ST_Collect(geom)) As geom
FROM l_shape) As foo;
```

wkt

```
POLYGON((6 194,6 190,14 194,6 194))
POLYGON((14 194,6 190,14 174,14 194))
POLYGON((14 194,14 174,154 14,14 194))
POLYGON((154 14,14 174,14 154,154 14))
POLYGON((154 14,14 154,150 14,154 14))
POLYGON((154 14,150 14,154 6,154 14))
```

Example using vertices with Z values.

3D multipoint

```
SELECT ST_AsText(ST_DelaunayTriangles(ST_GeomFromText(
'MULTIPOINT Z(14 14 10, 150 14 100,34 6 25, 20 10 150)')))) As wkt;
```

wkt

```
GEOMETRYCOLLECTION Z (POLYGON Z ((14 14 10,20 10 150,34 6 25,14 14 10))
,POLYGON Z ((14 14 10,34 6 25,150 14 100,14 14 10)))
```

☒☒

[ST_VoronoiPolygons](#), [ST_TriangulatePolygon](#), [ST_ConstrainedDelaunayTriangles](#), [ST_VoronoiLines](#), [ST_Con](#)

7.14.8 ST_FilterByM

ST_FilterByM — Removes vertices based on their M value

Synopsis

geometry **ST_FilterByM**(geometry geom, double precision min, double precision max = null, boolean returnM = false);

☒☒

Filters out vertex points based on their M-value. Returns a geometry with only vertex points that have a M-value larger or equal to the min value and smaller or equal to the max value. If max-value argument is left out only min value is considered. If fourth argument is left out the m-value will not be in the resulting geometry. If resulting geometry have too few vertex points left for its geometry type an empty geometry will be returned. In a geometry collection geometries without enough points will just be left out silently.

This function is mainly intended to be used in conjunction with ST_SetEffectiveArea. ST_EffectiveArea sets the effective area of a vertex in its m-value. With ST_FilterByM it then is possible to get a simplified version of the geometry without any calculations, just by filtering



Note

There is a difference in what ST_SimplifyVW returns when not enough points meet the criteria compared to ST_FilterByM. ST_SimplifyVW returns the geometry with enough points while ST_FilterByM returns an empty geometry



Note

Note that the returned geometry might be invalid



Note

This function returns all dimensions, including the Z and M values

Availability: 2.5.0

☐☐

A linestring is filtered

```
SELECT ST_AsText(ST_FilterByM(geom,30)) simplified
FROM (SELECT ST_SetEffectiveArea('LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry) geom ↵
      ) As foo;

result

          simplified
-----
LINESTRING(5 2,7 25,10 10)
```

☐☐

[ST_SetEffectiveArea](#), [ST_SimplifyVW](#)

7.14.9 ST_GeneratePoints

ST_GeneratePoints — Generates a multipoint of random points contained in a Polygon or MultiPolygon.

Synopsis

geometry **ST_GeneratePoints**(geometry g, integer npoints, integer seed = 0);

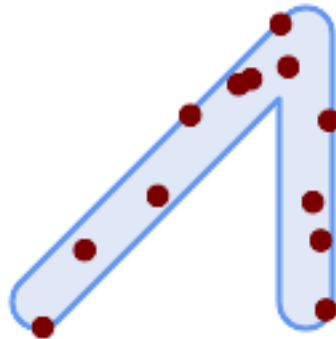
☐☐

ST_GeneratePoints generates a multipoint consisting of a given number of pseudo-random points which lie within the input area. The optional seed is used to regenerate a deterministic sequence of points, and must be greater than zero.

2.3.0 简体中文版。

Enhanced: 3.0.0, added seed parameter

图 7.14.10



Generated a multipoint consisting of 12 Points overlaid on top of original polygon using a random seed value 1996

```
SELECT ST_GeneratePoints(geom, 12, 1996)
FROM (
  SELECT ST_Buffer(
    ST_GeomFromText(
      'LINESTRING(50 50,150 150,150 50)'),
    10, 'endcap=round join=round') AS geom
) AS s;
```

Given a table of polygons *s*, return 12 individual points per polygon. Results will be different each time you run.

```
SELECT s.id, dp.path[1] AS pt_id, dp.geom
FROM s, ST_DumpPoints(ST_GeneratePoints(s.geom,12)) AS dp;
```

图 7.14.11

ST_DumpPoints

7.14.10 ST_GeometricMedian

ST_GeometricMedian — 返回几何体的几何中位数 (median) 几何体。

Synopsis

geometry **ST_GeometricMedian** (geometry geom, float8 tolerance = NULL, int max_iter = 10000, boolean fail_if_not_converged = false);

几何中位数

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.

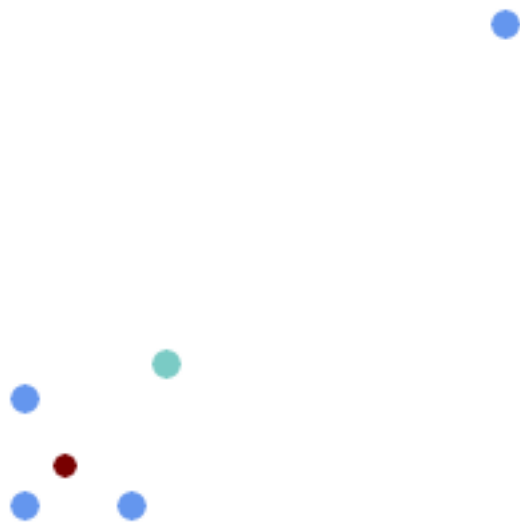
If present, the input point M values are interpreted as their relative weights.

2.3.0 简体中文

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.

几何中位数



Comparison of the geometric median (red) and centroid (turquoise) of a MultiPoint.

```
WITH test AS (
SELECT 'MULTIPOINT((10 10), (10 40), (40 10), (190 190))'::geometry geom)
SELECT
  ST_AsText(ST_Centroid(geom)) centroid,
  ST_AsText(ST_GeometricMedian(geom)) median
FROM test;
```

centroid	median
POINT(62.5 62.5)	POINT(25.01778421249728 25.01778421249728)

(1 row)

☒☒

ST_Centroid

7.14.11 ST_LineMerge

ST_LineMerge — Return the lines formed by sewing together a MultiLineString.

Synopsis

```
geometry ST_LineMerge(geometry amultilinestring);
geometry ST_LineMerge(geometry amultilinestring, boolean directed);
```

☒☒

Returns a LineString or MultiLineString formed by joining together the line elements of a MultiLineString. Lines are joined at their endpoints at 2-way intersections. Lines are not joined across intersections of 3-way or greater degree.

If **directed** is TRUE, then ST_LineMerge will not change point order within LineStrings, so lines with opposite directions will not be merged



Note

Only use with MultiLineString/LineStrings. Other geometry types return an empty GeometryCollection

GEOS 简体中文

Enhanced: 3.3.0 accept a directed parameter.

Requires GEOS >= 3.11.0 to use the directed parameter.

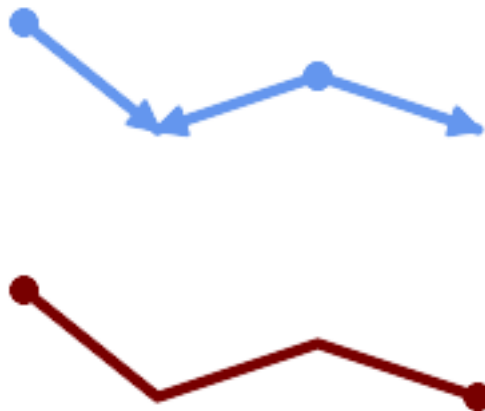
1.1.0 简体中文.



Warning

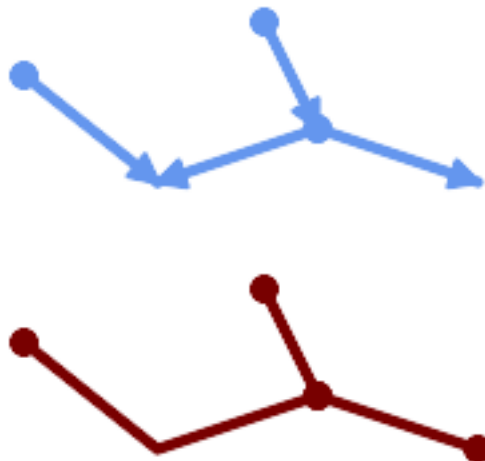
This function strips the M dimension.

图 10



Merging lines with different orientation.

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120))'
));
-----
LINESTRING(10 160,60 120,120 140,180 120)
```



Lines are not merged across intersections with degree > 2.

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120), (100 180, 120 140))'
));
-----
MULTILINESTRING((10 160,60 120,120 140),(100 180,120 140),(120 140,180 120))
```

If merging is not possible due to non-touching lines, the original MultiLineString is returned.

ST_MaximumInscribedCircle

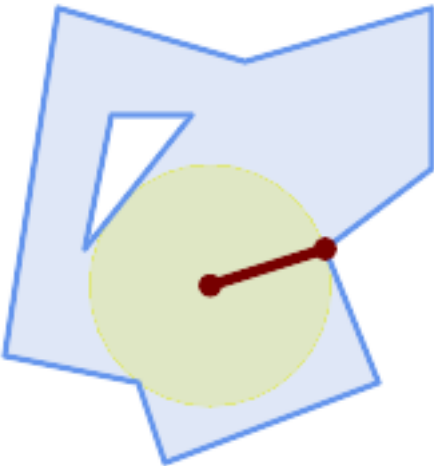
Finds the largest circle that is contained within a (multi)polygon, or which does not overlap any lines and points. Returns a record with fields:

- center - center point of the circle
- nearest - a point on the geometry nearest to the center
- radius - radius of the circle

For polygonal inputs, the circle is inscribed within the boundary rings, using the internal rings as boundaries. For linear and point inputs, the circle is inscribed within the convex hull of the input, using the input lines and points as further boundaries.

Availability: 3.1.0.
Requires GEOS >= 3.9.0.

描述

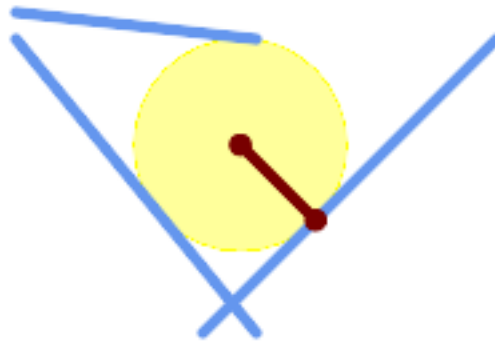


Maximum inscribed circle of a polygon. Center, nearest point, and radius are returned.

```
SELECT radius, ST_AsText(center) AS center, ST_AsText(nearest) AS nearest
FROM ST_MaximumInscribedCircle(
  'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 50, 40 180), (60 140, 50 90, 90 140, 60 140))');

```

radius	center	nearest
45.165845650018	POINT(96.953125 76.328125)	POINT(140 90)



Maximum inscribed circle of a multi-linestring. Center, nearest point, and radius are returned.

☒☒

[ST_MinimumBoundingRadius](#), [ST_LargestEmptyCircle](#)

7.14.13 ST_LargestEmptyCircle

`ST_LargestEmptyCircle` — Computes the largest circle not overlapping a geometry.

Synopsis

(geometry, geometry, double precision) **ST_LargestEmptyCircle**(geometry geom, double precision tolerance=0.0, geometry boundary=POINT EMPTY);

☒☒

Finds the largest circle which does not overlap a set of point and line obstacles. (Polygonal geometries may be included as obstacles, but only their boundary lines are used.) The center of the circle is constrained to lie inside a polygonal boundary, which by default is the convex hull of the input geometry. The circle center is the point in the interior of the boundary which has the farthest distance from the obstacles. The circle itself is provided by the center point and a nearest point lying on an obstacle determining the circle radius.

The circle center is determined to a given accuracy specified by a distance tolerance, using an iterative algorithm. If the accuracy distance is not specified a reasonable default is used.

Returns a record with fields:

- center - center point of the circle
- nearest - a point on the geometry nearest to the center
- radius - radius of the circle

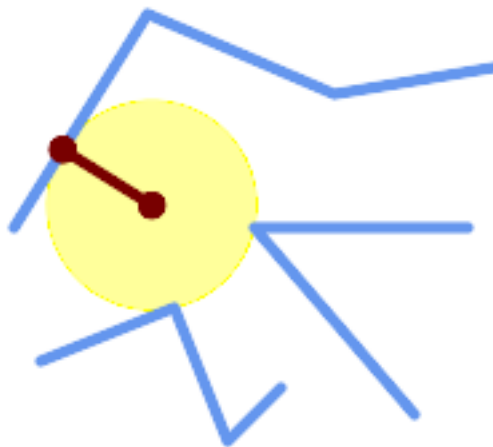
To find the largest empty circle in the interior of a polygon, see [ST_MaximumInscribedCircle](#).

Availability: 3.4.0.

Requires GEOS >= 3.9.0.

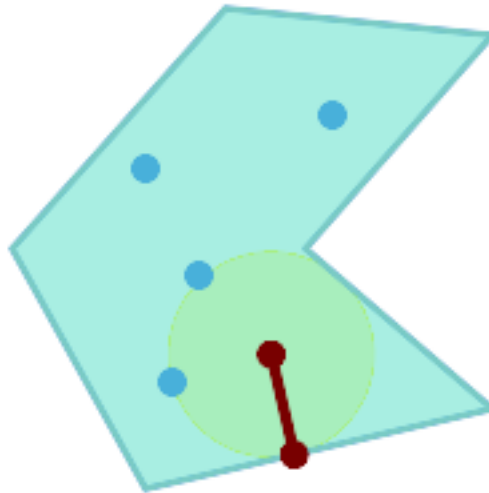
图

```
SELECT radius,
       center,
       nearest
FROM ST_LargestEmptyCircle(
    'MULTILINESTRING (
        (10 100, 60 180, 130 150, 190 160),
        (20 50, 70 70, 90 20, 110 40),
        (160 30, 100 100, 180 100))');
```



Largest Empty Circle within a set of lines.

```
SELECT radius,
       center,
       nearest
FROM ST_LargestEmptyCircle(
    ST_Collect(
        'MULTIPOINT ((70 50), (60 130), (130 150), (80 90))'::geometry,
        'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'::geometry) ←
        ,
        'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'::geometry
    );
```



Largest Empty Circle within a set of points, constrained to lie in a polygon. The constraint polygon boundary must be included as an obstacle, as well as specified as the constraint for the circle center.

ST_MinimumBoundingRadius

7.14.14 ST_MinimumBoundingCircle

ST_MinimumBoundingCircle — Returns the smallest circle polygon that contains a geometry.

Synopsis

```
geometry ST_MinimumBoundingCircle(geometry geomA, integer num_segs_per_qt_circ=48);
```



Returns the smallest circle polygon that contains a geometry.

Note



Note!

[illegible]

Use with **ST_Collect** to get the minimum bounding circle of a set of geometries.

To compute two points lying on the minimum circle (the "maximum diameter") use **ST_LongestLine**.

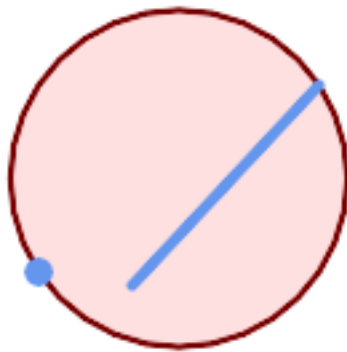
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (Roeck) XXXXXXXXXXXXXXXX.

GEOS ☐☐☐☐☐

1.4.0 ☒☒☒☒☒☒☒☒☒☒☒.

实验

```
SELECT d.disease_type,
       ST_MinimumBoundingCircle(ST_Collect(d.geom)) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```



实验四：空间数据库中的几何操作。实验四：空间数据库中的几何操作 8 实验四：空间数据库中的几何操作。

```
SELECT ST_AsText(ST_MinimumBoundingCircle(
    ST_Collect(
        ST_GeomFromText('LINESTRING(55 75,125 150)'),
        ST_Point(20, 80)), 8
    )) As wktmbc;

wktmbc
-----
POLYGON((135.59714732062 115,134.384753327498 102.690357210921,130.79416296937 ↵
  90.8537670908995,124.963360620072 79.9451031602111,117.116420743937 ↵
  70.3835792560632,107.554896839789 62.5366393799277,96.6462329091006 ↵
  56.70583703063,84.8096427890789 53.115246672502,72.5000000000001 ↵
  51.9028526793802,60.1903572109213 53.1152466725019,48.3537670908996 ↵
  56.7058370306299,37.4451031602112 62.5366393799276,27.8835792560632 ↵
  70.383579256063,20.0366393799278 79.9451031602109,14.20583703063 ↵
  90.8537670908993,10.615246672502 102.690357210921,9.40285267938019 115,10.6152466725019 ↵
  127.309642789079,14.2058370306299 139.1462329091,20.0366393799275 ↵
  150.054896839789,27.883579256063 159.616420743937,
  37.4451031602108 167.463360620072,48.3537670908992 173.29416296937,60.190357210921 ↵
  176.884753327498,
  72.4999999999998 178.09714732062,84.8096427890786 176.884753327498,96.6462329091003 ↵
  173.29416296937,107.554896839789 167.463360620072,
  117.116420743937 159.616420743937,124.963360620072 150.054896839789,130.79416296937 ↵
  139.146232909101,134.384753327498 127.309642789079,135.59714732062 115))
```

实验

ST_Collect, ST_MinimumBoundingRadius, ST_LargestEmptyCircle, ST_LongestLine

7.14.15 ST_MinimumBoundingRadius

ST_MinimumBoundingRadius — Returns the center point and radius of the smallest circle that contains a geometry.

Synopsis

(geometry, double precision) **ST_MinimumBoundingRadius**(geometry geom);

返回

Computes the center point and radius of the smallest circle that contains a geometry. Returns a record with fields:

- center - center point of the circle
- radius - radius of the circle

Use with **ST_Collect** to get the minimum bounding circle of a set of geometries.

To compute two points lying on the minimum circle (the "maximum diameter") use **ST_LongestLine**.

2.3.0 函数参考。

返回

```
SELECT ST_AsText(center), radius FROM ST_MinimumBoundingRadius('POLYGON((26426 65078,26531 65242,26075 65136,26096 65427,26426 65078))');
```

st_astext	radius
POINT(26284.8418027133 65267.1145090825)	247.436045591407

返回

ST_Collect, **ST_MinimumBoundingCircle**, **ST_LongestLine**

7.14.16 ST_OrientedEnvelope

ST_OrientedEnvelope — Returns a minimum-area rectangle containing a geometry.

Synopsis

geometry **ST_OrientedEnvelope**(geometry geom);

返回

Returns the minimum-area rotated rectangle enclosing a geometry. Note that more than one such rectangle may exist. May return a Point or LineString in the case of degenerate inputs.

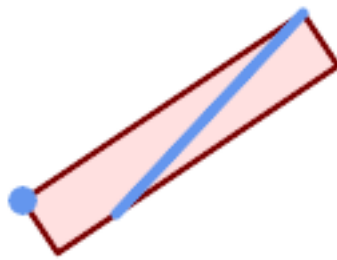
Availability: 2.5.0.

Requires GEOS >= 3.6.0.

图例

```
SELECT ST_AsText(ST_OrientedEnvelope('MULTIPOINT ((0 0), (-1 -1), (3 2))'));

st_astext
-----
POLYGON((3 2,2.88 2.16,-1.12 -0.84,-1 -1,3 2))
```



Oriented envelope of a point and linestring.

```
SELECT ST_AsText(ST_OrientedEnvelope(
    ST_Collect(
        ST_GeomFromText('LINESTRING(55 75,125 150)'),
        ST_Point(20, 80))
    )) As wktenv;

wktenv
-----
POLYGON((19.9999999999997 79.9999999999999,33.0769230769229 ↵
60.3846153846152,138.076923076924 130.384615384616,125.000000000001 ↵
150.000000000001,19.9999999999997 79.9999999999999))
```

图例

ST_Envelope ST_MinimumBoundingCircle

7.14.17 ST_OffsetCurve

ST_OffsetCurve — Returns an offset line at a given distance and side from an input line.

Synopsis

geometry **ST_OffsetCurve**(geometry line, float signed_distance, text style_parameters=“);

ST_OffsetCurve

Return an offset line at a given distance and side from an input line. All points of the returned geometries are not further than the given distance from the input geometry. Useful for computing parallel lines about a center line.

For positive distance the offset is on the left side of the input line and retains the same direction. For a negative distance it is on the right side and in the opposite direction.

Example: `ST_OffsetCurve('LINESTRING(0 0, 1 0, 1 1, 0 1, 0 0)', 0.5)`

Note that output may be a MULTILINESTRING or EMPTY for some jigsaw-shaped input geometries.

Example: `ST_OffsetCurve('MULTILINESTRING((0 0, 1 0, 1 1, 0 1, 0 0), (0.5 0.5, 1.5 0.5, 1.5 1.5, 0.5 1.5, 0.5 0.5))', 0.5)` = `ST_Multi(ST_OffsetCurve('MULTILINESTRING((0 0, 1 0, 1 1, 0 1, 0 0), (0.5 0.5, 1.5 0.5, 1.5 1.5, 0.5 1.5, 0.5 0.5))', 0.5))`

- 'quad_segs=#': integer (quarter circle) segments (default 8)
- 'join=round|mitre|bevel': string ('round' | 'mitre' | 'bevel'). 'mitre' requires 'mitre_limit' parameter.
- 'mitre_limit=#.#': float (mitre limit). 'mitre_limit' requires 'miter_limit' parameter.

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 Example: `ST_OffsetCurve('LINESTRING(0 0, 1 0, 1 1, 0 1, 0 0)', 0.5)`

Enhanced: 2.5 - added support for GEOMETRYCOLLECTION and MULTILINESTRING



Note

This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry.

ST_MakeEnvelope

Example: `ST_MakeEnvelope(0, 1, 0, 1, 0.5)`

```
SELECT ST_Union(
  ST_OffsetCurve(f.geom, f.width/2, 'quad_segs=4 join=round'),
  ST_OffsetCurve(f.geom, -f.width/2, 'quad_segs=4 join=round')
) as track
FROM someroadstable;
```



国国 15, 国国国国国'quad_segs=4 join=round'
国国国国 15 国国

```
SELECT ST_AsText(ST_OffsetCurve(↵
    ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104 ↵
    16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)')↵
    ,
    15, 'quad_segs=4 join=round'));
```

output

```
LINESTRING(164 1,18 1,12.2597485145237↵
    2.1418070123307,
    7.39339828220179 5.39339828220179,
    5.39339828220179 7.39339828220179,
    2.14180701233067 12.2597485145237,1↵
    18,1 195)
```

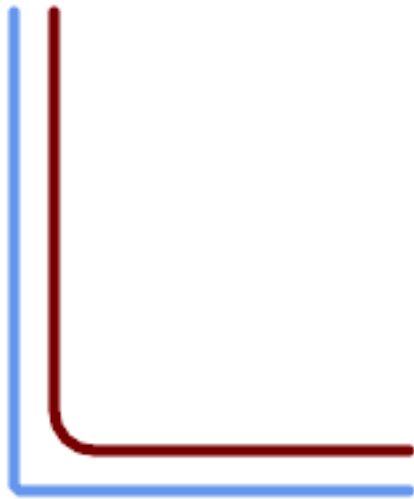


国国 -15, 国国国国国'quad_segs=4
join=round' 国国国国 -15 国国

```
SELECT ST_AsText(ST_OffsetCurve(geom,↵
    -15, 'quad_segs=4 join=round')) As↵
    notsocurvy
FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104↵
    16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)')↵
    As geom;
```

notsocurvy

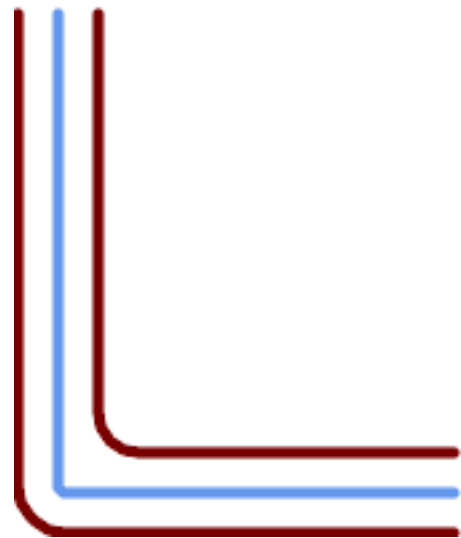
```
LINESTRING(31 195,31 31,164 31)
```



图例显示了蓝色和红色的 L 形线。红色线是蓝色线向外偏移 15 个单位的结果，而蓝色线是红色线向内偏移 15 个单位的结果。偏移距离为 -30 + 15 = -15 个单位。

```
SELECT ST_AsText(ST_OffsetCurve(↵
    ST_OffsetCurve(geom,↵
        -30, 'quad_segs=4 join=round'), -15, ↵
        'quad_segs=4 join=round')) As morecurvy
FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104 ↵
    16,84 16,64 16,↵
    44 16,24 16,20 16,18 16,17 17,↵
    16 18,16 20,16 40,16 60,16 80,16 100,↵
    16 120,16 140,16 160,16 180,16 195)') ↵
    As geom;
morecurvy
```

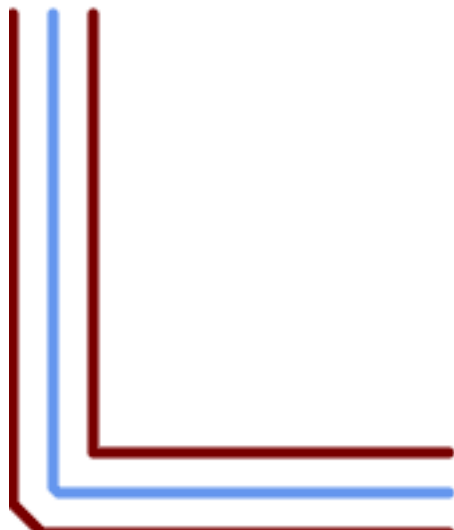
```
LINESTRING(164 31,46 31,40.2597485145236 ↵
    32.1418070123307,↵
    35.3933982822018 35.3933982822018,↵
    32.1418070123307 40.2597485145237,31 ↵
    46,31 195)
```



图例显示了蓝色和红色的 L 形线。红色线是蓝色线向外偏移 15 个单位的结果，而蓝色线是红色线向内偏移 15 个单位的结果。偏移距离为 -30 + 15 = -15 个单位。

```
SELECT ST_AsText(ST_Collect(↵
    ST_OffsetCurve(geom, 15, 'quad_segs=4 ↵
        join=round'),↵
    ST_OffsetCurve(ST_OffsetCurve(geom,↵
        -30, 'quad_segs=4 join=round'), -15, ↵
        'quad_segs=4 join=round')↵
    ) As parallel_curves
FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104 ↵
    16,84 16,64 16,↵
    44 16,24 16,20 16,18 16,17 17,↵
    16 18,16 20,16 40,16 60,16 80,16 100,↵
    16 120,16 140,16 160,16 180,16 195)') ↵
    As geom;
parallel curves

MULTILINESTRING((164 1,18 ↵
    1,12.2597485145237 2.1418070123307,↵
    7.39339828220179 ↵
    5.39339828220179,5.39339828220179 7.39339828220179,↵
    2.14180701233067 12.2597485145237,1 18,1 ↵
    195),↵
    (164 31,46 31,40.2597485145236 ↵
    32.1418070123307,35.3933982822018 35.3933982822018,↵
    32.1418070123307 40.2597485145237,31 ↵
    46,31 195))
```


 15, 'quad_segs=4 join=round' <pre>SELECT ST_AsText(ST_OffsetCurve(↵ ST_GeomFromText(↵ 'LINESTRING(164 16,144 16,124 16,104 ↵ 16,84 16,64 16,↵ 44 16,24 16,20 16,18 16,17 17,↵ 16 18,16 20,16 40,16 60,16 80,16 100,↵ 16 120,16 140,16 160,16 180,16 195)')↵ ,↵ 15, 'quad_segs=4 join=bevel'));</pre> output <pre>LINESTRING(164 1,18 1,7.39339828220179 ↵ 5.39339828220179,↵ 5.39339828220179 7.39339828220179,1 ↵ 18,1 195)</pre>	 15, -15. join=mitre mitre_limit=2.1 <pre>SELECT ST_AsText(ST_Collect(↵ ST_OffsetCurve(geom, 15, 'quad_segs=4 ↵ join=mitre mitre_limit=2.2'),↵ ST_OffsetCurve(geom, -15, 'quad_segs ↵ =4 join=mitre mitre_limit=2.2')↵))↵ FROM ST_GeomFromText(↵ 'LINESTRING(164 16,144 16,124 16,104 ↵ 16,84 16,64 16,↵ 44 16,24 16,20 16,18 16,17 17,↵ 16 18,16 20,16 40,16 60,16 80,16 100,↵ 16 120,16 140,16 160,16 180,16 195)')↵ As geom;</pre> output <pre>MULTILINESTRING((164 1,11.7867965644036 ↵ 1,1 11.7867965644036,1 195),↵ (31 195,31 31,164 31))</pre>

图例

ST_Buffer

7.14.18 ST_PointOnSurface

ST_PointOnSurface — Computes a point guaranteed to lie in a polygon, or on a geometry.

Synopsis

geometry **ST_PointOnSurface**(geometry g1);



Returns a `POINT` which is guaranteed to lie in the interior of a surface (`POLYGON`, `MULTIPOLYGON`, and `CURVEPOLYGON`). In PostGIS this function also works on line and point geometries.



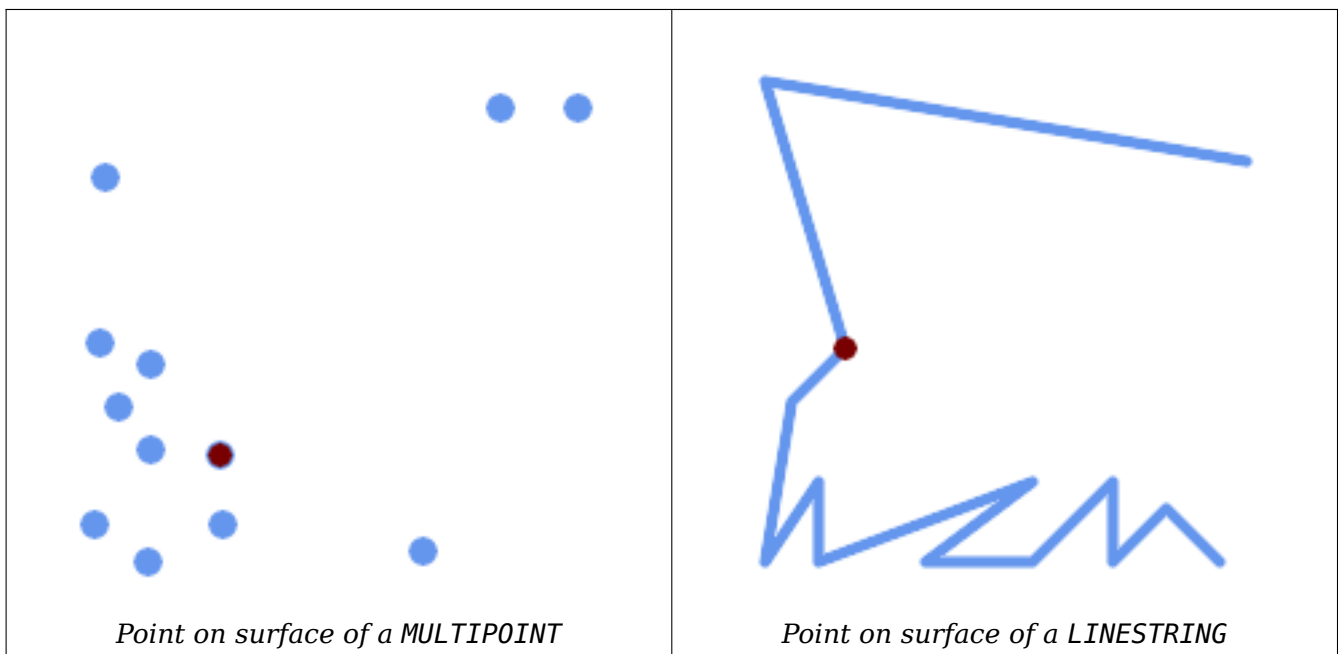
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.14.2 // s3.2.18.2

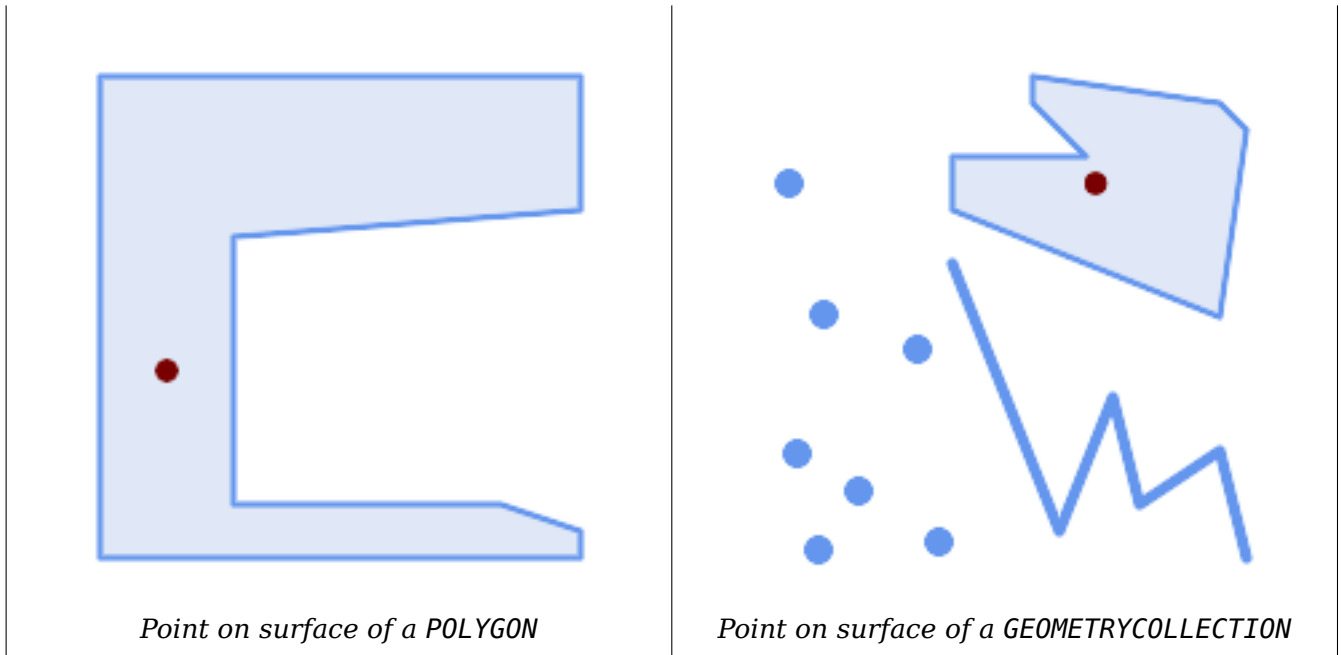


This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. The specifications define `ST_PointOnSurface` for surface geometries only. PostGIS extends the function to support all common geometry types. Other databases (Oracle, DB2, ArcSDE) seem to support this function only for surfaces. SQL Server 2008 supports all common geometry types.



This function supports 3d and will not drop the z-index.





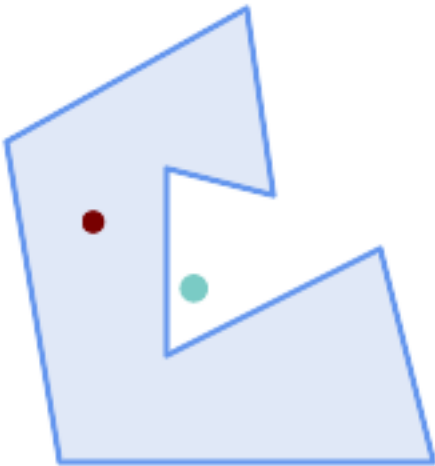
```
SELECT ST_AsText(ST_PointOnSurface('POINT(0 5)')::geometry);
-----
POINT(0 5)

SELECT ST_AsText(ST_PointOnSurface('LINESTRING(0 5, 0 10)')::geometry);
-----
POINT(0 5)

SELECT ST_AsText(ST_PointOnSurface('POLYGON((0 0, 0 5, 5 5, 5 0, 0 0))')::geometry);
-----
POINT(2.5 2.5)

SELECT ST_AsEWKT(ST_PointOnSurface(ST_GeomFromEWKT('LINESTRING(0 5 1, 0 0 1, 0 10 2)')));
-----
POINT(0 0 1)
```

Example: The result of `ST_PointOnSurface` is guaranteed to lie within polygons, whereas the point computed by `ST_Centroid` may be outside.



Red: point on surface; Green: centroid

```
SELECT ST_AsText(ST_PointOnSurface(geom)) AS pt_on_surf,
       ST_AsText(ST_Centroid(geom)) AS centroid
FROM (SELECT 'POLYGON ((130 120, 120 190, 30 140, 50 20, 190 20,
                        170 100, 90 60, 90 130, 130 120))'::geometry AS geom) AS t;
```

pt_on_surf	centroid
POINT(62.5 110)	POINT(100.18264840182648 85.11415525114155)

🔗🔗

[ST_Centroid](#), [ST_MaximumInscribedCircle](#)

7.14.19 ST_Polygonize

ST_Polygonize — Computes a collection of polygons formed from the linework of a set of geometries.

Synopsis

geometry **ST_Polygonize**(geometry set geomfield);
geometry **ST_Polygonize**(geometry[] geom_array);

🔗🔗

Creates a GeometryCollection containing the polygons formed by the linework of a set of geometries. If the input linework does not form any polygons, an empty GeometryCollection is returned.

This function creates polygons covering all delimited areas. If the result is intended to form a valid polygonal geometry, use [ST_BuildArea](#) to prevent holes being filled.



Note
The input linework must be correctly noded for this function to work properly. To ensure input is noded use [ST_Node](#) on the input geometry before polygonizing.

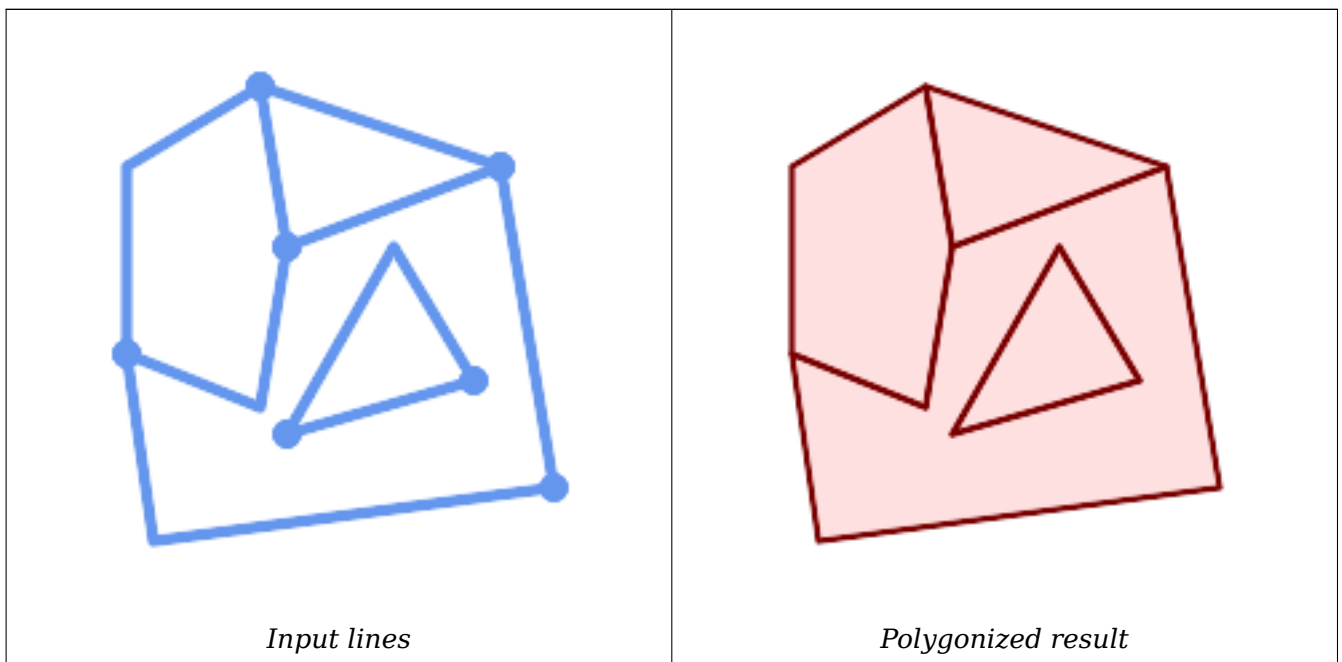
**Note**

GeometryCollections can be difficult to handle with external tools. Use **ST_Dump** to convert the polygonized result into separate polygons.

GEOS 3.10.2

1.0.0RC1 简体中文版.

图



```
WITH data(geom) AS (VALUES
  ('LINESTRING (180 40, 30 20, 20 90)'::geometry)
, ('LINESTRING (180 40, 160 160)'::geometry)
, ('LINESTRING (80 60, 120 130, 150 80)'::geometry)
, ('LINESTRING (80 60, 150 80)'::geometry)
, ('LINESTRING (20 90, 70 70, 80 130)'::geometry)
, ('LINESTRING (80 130, 160 160)'::geometry)
, ('LINESTRING (20 90, 20 160, 70 190)'::geometry)
, ('LINESTRING (70 190, 80 130)'::geometry)
, ('LINESTRING (70 190, 160 160)'::geometry)
)
SELECT ST_AsText( ST_Polygonize( geom ))
FROM data;
```

```
-----
GEOMETRYCOLLECTION (POLYGON ((180 40, 30 20, 20 90, 70 70, 80 130, 160 160, 180 40), (150 80, 120 130, 80 60, 150 80)),
  POLYGON ((20 90, 20 160, 70 190, 80 130, 70 70, 20 90)),
  POLYGON ((160 160, 80 130, 70 190, 160 160)),
  POLYGON ((80 60, 120 130, 150 80, 80 60)))
```

Polygonizing a table of linestrings:

```

SELECT ST_AsEWKT(ST_Polygonize(geom_4269)) As geomtextrep
FROM (SELECT geom_4269 FROM ma.suffolk_edges) As foo;

-----
SRID=4269;GEOMETRYCOLLECTION(POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 42.285752,-71.040878 42.285678)),
POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,-71.171794 42.354971,-71.170511 42.354855,
-71.17112 42.354238,-71.17166 42.353675)))

--Use ST_Dump to dump out the polygonize geoms into individual polygons
SELECT ST_AsEWKT((ST_Dump(t.polycoll)).geom) AS geomtextrep
FROM (SELECT ST_Polygonize(geom_4269) AS polycoll
      FROM (SELECT geom_4269 FROM ma.suffolk_edges)
      As foo) AS t;

-----
SRID=4269;POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 42.285752,-71.040878 42.285678))
SRID=4269;POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,-71.171794 42.354971,-71.170511 42.354855,-71.17112 42.354238,-71.17166 42.353675))

```

☒☒

[ST_BuildArea](#), [ST_Dump](#), [ST_Node](#)

7.14.20 ST_ReducePrecision

ST_ReducePrecision — Returns a valid geometry with points rounded to a grid tolerance.

Synopsis

geometry **ST_ReducePrecision**(geometry g, float8 gridsz);

☒☒

Returns a valid geometry with all points rounded to the provided grid tolerance, and features below the tolerance removed.

Unlike [ST_SnapToGrid](#) the returned geometry will be valid, with no ring self-intersections or collapsed components.

Precision reduction can be used to:

- match coordinate precision to the data accuracy
- reduce the number of coordinates needed to represent a geometry
- ensure valid geometry output to formats which use lower precision (e.g. text formats such as WKT, GeoJSON or KML when the number of output decimal places is limited).
- export valid geometry to systems which use lower or limited precision (e.g. SDE, Oracle tolerance value)

Availability: 3.1.0.

Requires GEOS >= 3.9.0.

例

```
SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 0.1));
      st_astext
-----
POINT(1.4 19.3)

SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 1.0));
      st_astext
-----
POINT(1 19)

SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 10));
      st_astext
-----
POINT(0 20)
```

Precision reduction can reduce number of vertices

```
SELECT ST_AsText(ST_ReducePrecision('LINESTRING (10 10, 19.6 30.1, 20 30, 20.3 30, 40 40)', 1));
      st_astext
-----
LINESTRING (10 10, 20 30, 40 40)
```

Precision reduction splits polygons if needed to ensure validity

```
SELECT ST_AsText(ST_ReducePrecision('POLYGON ((10 10, 60 60.1, 70 30, 40 40, 50 10, 10 10))', 10));
      st_astext
-----
MULTIPOLYGON (((60 60, 70 30, 40 40, 60 60)), ((40 40, 50 10, 10 10, 40 40)))
```

例

ST_SnapToGrid, **ST_Simplify**, **ST_SimplifyVW**

7.14.21 ST_SharedPaths

ST_SharedPaths — 返回两个几何体共享的路径。

Synopsis

geometry **ST_SharedPaths**(geometry lineal1, geometry lineal2);

例

```
SELECT ST_AsText(ST_SharedPaths('LINESTRING (0 0, 10 10, 20 10, 30 10, 40 10, 50 10, 60 10, 70 10, 80 10, 90 10, 100 10)', 'LINESTRING (0 0, 10 10, 20 10, 30 10, 40 10, 50 10, 60 10, 70 10, 80 10, 90 10, 100 10)'));
      st_astext
-----
LINESTRING (0 0, 10 10, 20 10, 30 10, 40 10, 50 10, 60 10, 70 10, 80 10, 90 10, 100 10)
```

GEOS 版本

2.0.0 版本。

例: 共享路径



图 10-10 共享路径



图 10-11 共享路径

```
SELECT ST_AsText(  
  ST_SharedPaths(  
    ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),  
      (51 150,101 150,76 175,51 150))'),  
    ST_GeomFromText('LINESTRING(151 100,126 156.25,126 125,90 161, 76 175)')  
  )  
  ) As wkt  
  
-----  
wkt  
-----  
GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125),  
  (101 150,90 161),(90 161,76 175)),MULTILINESTRING EMPTY)
```


same example but linestring orientation flipped

```
SELECT ST_AsText(
  ST_SharedPaths(
    ST_GeomFromText('LINESTRING(76 175,90 161,126 125,126 156.25,151 100)'),
    ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
      (51 150,101 150,76 175,51 150))')
  )
) As wkt
```

wkt

```
-----
GEOMETRYCOLLECTION(MULTILINESTRING EMPTY,
MULTILINESTRING((76 175,90 161),(90 161,101 150),(126 125,126 156.25)))
```

￼

[ST_Dump](#), [ST_GeometryN](#), [ST_NumGeometries](#)

7.14.22 ST_Simplify

ST_Simplify — Returns a simplified representation of a geometry, using the Douglas-Peucker algorithm.

Synopsis

```
geometry ST_Simplify(geometry geom, float tolerance);
geometry ST_Simplify(geometry geom, float tolerance, boolean preserveCollapsed);
```

￼

Computes a simplified representation of a geometry using the [Douglas-Peucker algorithm](#). The simplification tolerance is a distance value, in the units of the input SRS. Simplification removes vertices which are within the tolerance distance of the simplified linework. The result may not be valid even if the input is.

The function can be called with any kind of geometry (including GeometryCollections), but only line and polygon elements are simplified. Endpoints of linear geometry are preserved.

The `preserveCollapsed` flag retains small geometries that would otherwise be removed at the given tolerance. For example, if a 1m long line is simplified with a 10m tolerance, when `preserveCollapsed` is true the line will not disappear. This flag is useful for rendering purposes, to prevent very small features disappearing from a map.



Note

The returned geometry may lose its simplicity (see [ST_IsSimple](#)), topology may not be preserved, and polygonal results may be invalid (see [ST_IsValid](#)). Use [ST_SimplifyPreserveTopology](#) to preserve topology and ensure validity.

**Note**

This function does not preserve boundaries shared between polygons. Use **ST_CoverageSimplify** if this is required.

1.2.2 简化几何图形。

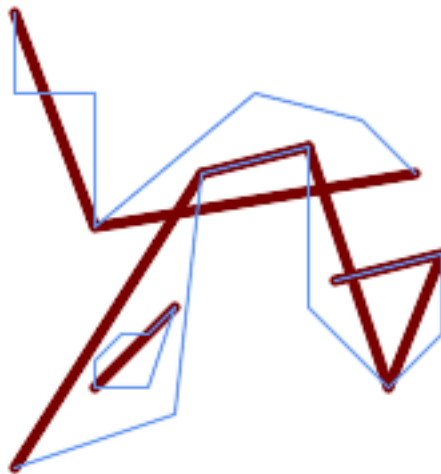
图 1.2.2

简化几何图形。

```
SELECT ST_Npoints(geom) AS np_before,
       ST_NPoints(ST_Simplify(geom, 0.1)) AS np01_notbadcircle,
       ST_NPoints(ST_Simplify(geom, 0.5)) AS np05_notquitecircle,
       ST_NPoints(ST_Simplify(geom, 1)) AS np1_octagon,
       ST_NPoints(ST_Simplify(geom, 10)) AS np10_triangle,
       (ST_Simplify(geom, 100) is null) AS np100_geometrygoesaway
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) As geom) AS t;
```

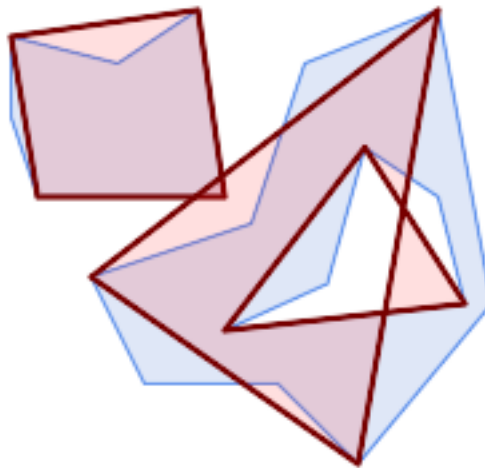
np_before	np01_notbadcircle	np05_notquitecircle	np1_octagon	np10_triangle	np100_geometrygoesaway
49	33	17	9	4	t

简化一组线条。线条可能在简化后相交。



```
SELECT ST_Simplify(
  'MULTILINESTRING ((20 180, 20 150, 50 150, 50 100, 110 150, 150 140, 170 120), (20 10, 80 10, 80 30, 90 120), (90 120, 130 130), (130 130, 130 70, 160 40, 180 60, 180 90, 140 80), (50 40, 70 40, 80 70, 70 60, 60 60, 50 50, 50 40))',
  40);
```

简化一个 MultiPolygon。多边形结果可能是无效的。



```
SELECT ST_Simplify(
  'MULTIPOLYGON (((90 110, 80 180, 50 160, 10 170, 10 140, 20 110, 90 110)), ((40 80, 100 80, 100 120 160, 170 180, 190 70, 140 10, 110 40, 60 40, 40 80), (180 70, 170 110, 142.5 128.5, 128.5 77.5, 90 60, 180 70)))',
  40);
```

☒☒

[ST_IsSimple](#), [ST_SimplifyPreserveTopology](#), [ST_SimplifyVW](#), [ST_CoverageSimplify](#), [Topology](#) [ST_Simplify](#)

7.14.23 ST_SimplifyPreserveTopology

ST_SimplifyPreserveTopology — Returns a simplified and valid representation of a geometry, using the Douglas-Peucker algorithm.

Synopsis

geometry **ST_SimplifyPreserveTopology**(geometry geom, float tolerance);

☒☒

Computes a simplified representation of a geometry using a variant of the [Douglas-Peucker algorithm](#) which limits simplification to ensure the result has the same topology as the input. The simplification tolerance is a distance value, in the units of the input SRS. Simplification removes vertices which are within the tolerance distance of the simplified linework, as long as topology is preserved. The result will be valid and simple if the input is.

The function can be called with any kind of geometry (including GeometryCollections), but only line and polygon elements are simplified. For polygonal inputs, the result will have the same number of rings (shells and holes), and the rings will not cross. Ring endpoints may be simplified. For linear inputs, the result will have the same number of lines, and lines will not intersect if they did not do so in the original geometry. Endpoints of linear geometry are preserved.



Note

This function does not preserve boundaries shared between polygons. Use **ST_CoverageSimplify** if this is required.

GEOS 函数

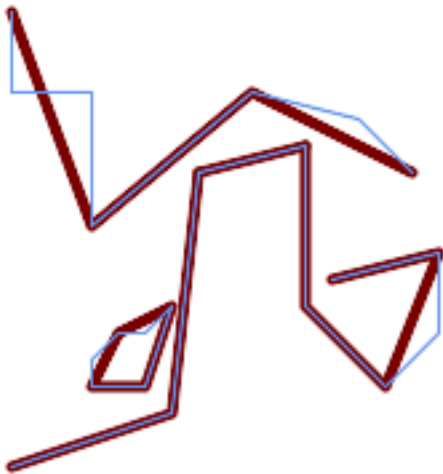
1.3.3 函数.

For the same example as **ST_Simplify**, **ST_SimplifyPreserveTopology** prevents oversimplification. The circle can at most become a square.

```
SELECT ST_Npoints geom AS np_before,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.1)) AS np01_notbadcircle,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.5)) AS np05_notquitecircle,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 1)) AS np1_octagon,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 10)) AS np10_square,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 100)) AS np100_stillsquare
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) AS geom) AS t;
```

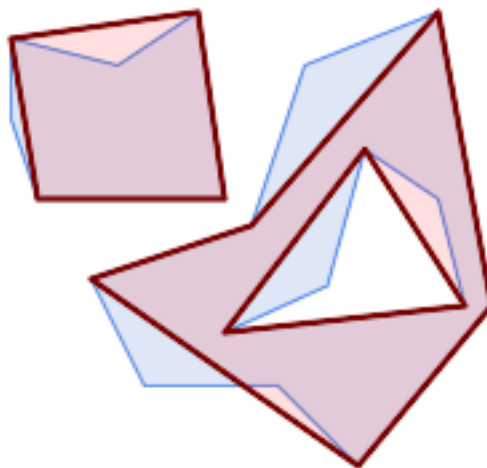
np_before	np01_notbadcircle	np05_notquitecircle	np1_octagon	np10_square	
49	33	17	9	5	
	5				

Simplifying a set of lines, preserving topology of non-intersecting lines.



```
SELECT ST_SimplifyPreserveTopology(
  'MULTILINESTRING ((20 180, 20 150, 50 150, 50 100, 110 150, 150 140, 170 120), (20 10, 80 10, 80 30, 90 120), (90 120, 130 130), (130 130, 130 70, 160 40, 180 60, 180 90, 140 80), (50 40, 70 40, 80 70, 70 60, 60 60, 50 50, 50 40))',
  40);
```

Simplifying a MultiPolygon, preserving topology of shells and holes.



```
SELECT ST_SimplifyPreserveTopology(
  'MULTIPOLYGON (((90 110, 80 180, 50 160, 10 170, 10 140, 20 110, 90 110)), ((40 80, 100 80, 100 120, 160 170, 180 190, 70 140, 10 110, 40 60, 40 40, 80 40), (180 70, 170 110, 142.5 128.5, 128.5 77.5, 90 60, 180 70)))',
  40);
```

☒☒

[ST_Simplify](#), [ST_SimplifyVW](#), [ST_CoverageSimplify](#)

7.14.24 ST_SimplifyPolygonHull

ST_SimplifyPolygonHull — Computes a simplified topology-preserving outer or inner hull of a polygonal geometry.

Synopsis

geometry **ST_SimplifyPolygonHull**(geometry param_geom, float vertex_fraction, boolean is_outer = true);

☒☒

Computes a simplified topology-preserving outer or inner hull of a polygonal geometry. An outer hull completely covers the input geometry. An inner hull is completely covered by the input geometry. The result is a polygonal geometry formed by a subset of the input vertices. MultiPolygons and holes are handled and produce a result with the same structure as the input.

The reduction in vertex count is controlled by the `vertex_fraction` parameter, which is a number in the range 0 to 1. Lower values produce simpler results, with smaller vertex count and less concaveness. For both outer and inner hulls a vertex fraction of 1.0 produces the original geometry. For outer hulls a value of 0.0 produces the convex hull (for a single polygon); for inner hulls it produces a triangle.

The simplification process operates by progressively removing concave corners that contain the least amount of area, until the vertex count target is reached. It prevents edges from crossing, so the result is always a valid polygonal geometry.

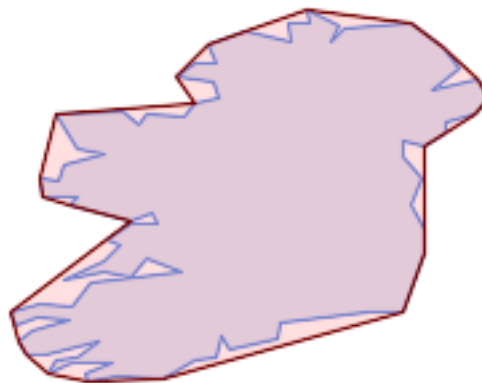
To get better results with geometries that contain relatively long line segments, it might be necessary to "segmentize" the input, as shown below.

GEOS 简体中文版

Availability: 3.3.0.

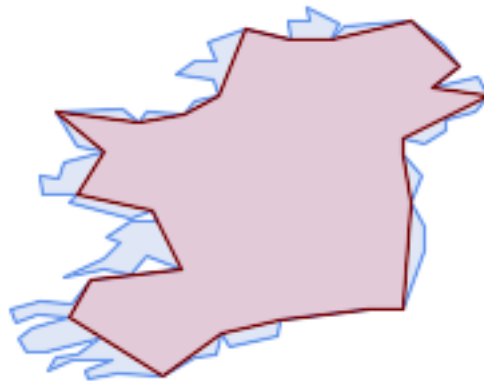
Requires GEOS >= 3.11.0.

图例



Outer hull of a Polygon

```
SELECT ST_SimplifyPolygonHull(
  'POLYGON ((131 158, 136 163, 161 165, 173 156, 179 148, 169 140, 186 144, 190 137, 185 ↵
    131, 174 128, 174 124, 166 119, 158 121, 158 115, 165 107, 161 97, 166 88, 166 79, 158 ↵
    57, 145 57, 112 53, 111 47, 93 43, 90 48, 88 40, 80 39, 68 32, 51 33, 40 31, 39 34, ↵
    49 38, 34 38, 25 34, 28 39, 36 40, 44 46, 24 41, 17 41, 14 46, 19 50, 33 54, 21 55, 13 ↵
    52, 11 57, 22 60, 34 59, 41 68, 75 72, 62 77, 56 70, 46 72, 31 69, 46 76, 52 82, 47 ↵
    84, 56 90, 66 90, 64 94, 56 91, 33 97, 36 100, 23 100, 22 107, 29 106, 31 112, 46 116, ↵
    36 118, 28 131, 53 132, 59 127, 62 131, 76 130, 80 135, 89 137, 87 143, 73 145, 80 ↵
    150, 88 150, 85 157, 99 162, 116 158, 115 165, 123 165, 122 170, 134 164, 131 158))',
  0.3);
```



Inner hull of a Polygon

```
SELECT ST_SimplifyPolygonHull(
  'POLYGON ((131 158, 136 163, 161 165, 173 156, 179 148, 169 140, 186 144, 190 137, 185 ↵
    131, 174 128, 174 124, 166 119, 158 121, 158 115, 165 107, 161 97, 166 88, 166 79, 158 ↵
    57, 145 57, 112 53, 111 47, 93 43, 90 48, 88 40, 80 39, 68 32, 51 33, 40 31, 39 34, ↵
    49 38, 34 38, 25 34, 28 39, 36 40, 44 46, 24 41, 17 41, 14 46, 19 50, 33 54, 21 55, 13 ↵
    52, 11 57, 22 60, 34 59, 41 68, 75 72, 62 77, 56 70, 46 72, 31 69, 46 76, 52 82, 47 ↵
    84, 56 90, 66 90, 64 94, 56 91, 33 97, 36 100, 23 100, 22 107, 29 106, 31 112, 46 116, ↵
    36 118, 28 131, 53 132, 59 127, 62 131, 76 130, 80 135, 89 137, 87 143, 73 145, 80 ↵
    150, 88 150, 85 157, 99 162, 116 158, 115 165, 123 165, 122 170, 134 164, 131 158))',
  0.3, false);
```



Outer hull simplification of a MultiPolygon, with segmentization

```
SELECT ST_SimplifyPolygonHull(
  ST_Segmentize(ST_Letters('xt'), 2.0),
  0.1);
```

☒☒

[ST_ConvexHull](#), [ST_SimplifyVW](#), [ST_ConcaveHull](#), [ST_Segmentize](#)

7.14.25 ST_SimplifyVW

ST_SimplifyVW — Returns a simplified representation of a geometry, using the Visvalingam-Whyatt algorithm

Synopsis

```
geometry ST_SimplifyVW(geometry geom, float tolerance);
```



Returns a simplified representation of a geometry using the **Visvalingam-Whyatt algorithm**. The simplification tolerance is an area value, in the units of the input SRS. Simplification removes vertices which form "corners" with area less than the tolerance. The result may not be valid even if the input is.

The function can be called with any kind of geometry (including GeometryCollections), but only line and polygon elements are simplified. Endpoints of linear geometry are preserved.

Note



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



Note!

This function does not preserve boundaries shared between polygons. Use `ST_CoverageSimplify` if this is required.

Note



Note!

Note

□ □ □ □ 3 □ □ □ □ □ □ □ , □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ .

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.

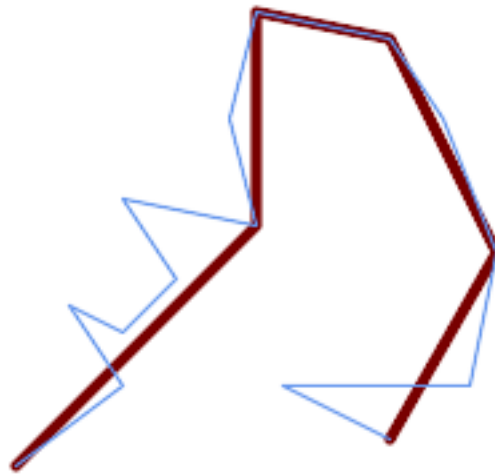


A LineString is simplified with a minimum-area tolerance of 30.

```
SELECT ST_AsText(ST_SimplifyVW(geom,30)) simplified
FROM (SELECT 'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)>::geometry AS geom) AS t;

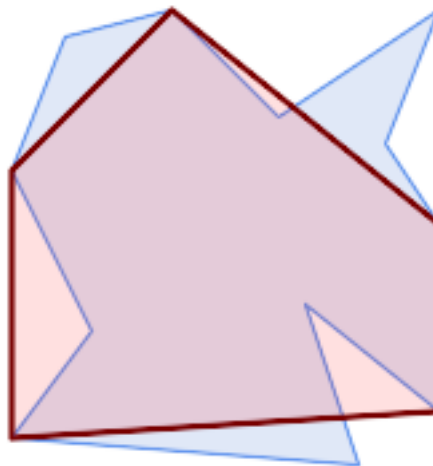
simplified
-----
LINESTRING(5 2,7 25,10 10)
```

Simplifying a line.



```
SELECT ST_SimplifyVW(
  'LINESTRING (10 10, 50 40, 30 70, 50 60, 70 80, 50 110, 100 100, 90 140, 100 180, 150 170, 170 140, 190 90, 180 40, 110 40, 150 20)',
  1600);
```

Simplifying a polygon.



```
SELECT ST_SimplifyVW(
  'MULTIPOLYGON (((90 110, 80 180, 50 160, 10 170, 10 140, 20 110, 90 110)), ((40 80, 100 100, 120 160, 170 180, 190 70, 140 10, 110 40, 60 40, 40 80), (180 70, 170 110, 142.5 128.5, 128.5 77.5, 90 60, 180 70))))',
  40);
```



[ST_SetEffectiveArea](#), [ST_Simplify](#), [ST_SimplifyPreserveTopology](#), [ST_CoverageSimplify](#), [Topology](#) [ST_Simpl](#)

7.14.26 ST_SetEffectiveArea

`ST_SetEffectiveArea` — Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.

☒☒

[ST_SimplifyVW](#)

7.14.27 ST_TriangulatePolygon

ST_TriangulatePolygon — Computes the constrained Delaunay triangulation of polygons

Synopsis

geometry **ST_TriangulatePolygon**(geometry geom);

☒☒

Computes the constrained Delaunay triangulation of polygons. Holes and Multipolygons are supported.

The “constrained Delaunay triangulation” of a polygon is a set of triangles formed from the vertices of the polygon, and covering it exactly, with the maximum total interior angle over all possible triangulations. It provides the “best quality” triangulation of the polygon.

Availability: 3.3.0.

Requires GEOS >= 3.11.0.

☒☒

Triangulation of a square.

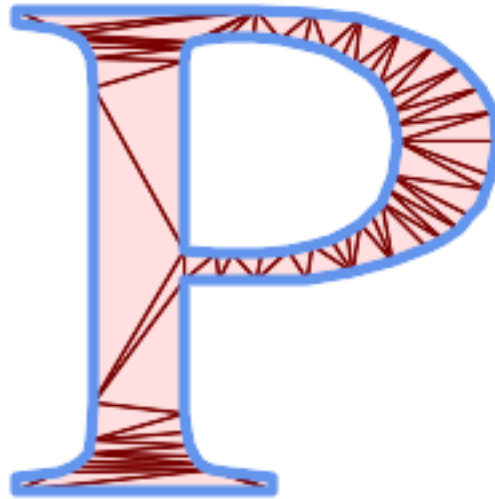
```
SELECT ST_AsText(
  ST_TriangulatePolygon('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))');

          st_astext
-----
GEOMETRYCOLLECTION(POLYGON((0 0,0 1,1 1,0 0)),POLYGON((1 1,1 0,0 0,1 1)))
```

☒☒

Triangulation of the letter P.

```
SELECT ST_AsText(ST_TriangulatePolygon(
  'POLYGON ((26 17, 31 19, 34 21, 37 24, 38 29, 39 43, 39 161, 38 172, 36 176, 34 179, 30 ↵
    181, 25 183, 10 185, 10 190, 100 190, 121 189, 139 187, 154 182, 167 177, 177 169, ↵
    184 161, 189 152, 190 141, 188 128, 186 123, 184 117, 180 113, 176 108, 170 104, 164 ↵
    101, 151 96, 136 92, 119 89, 100 89, 86 89, 73 89, 73 39, 74 32, 75 27, 77 23, 79 ↵
    20, 83 18, 89 17, 106 15, 106 10, 10 10, 10 15, 26 17), (152 147, 151 152, 149 157, ↵
    146 162, 142 166, 137 169, 132 172, 126 175, 118 177, 109 179, 99 180, 89 180, 80 ↵
    179, 76 178, 74 176, 73 171, 73 100, 85 99, 91 99, 102 99, 112 100, 121 102, 128 ↵
    104, 134 107, 139 110, 143 114, 147 118, 149 123, 151 128, 153 141, 152 147)))'
));
```



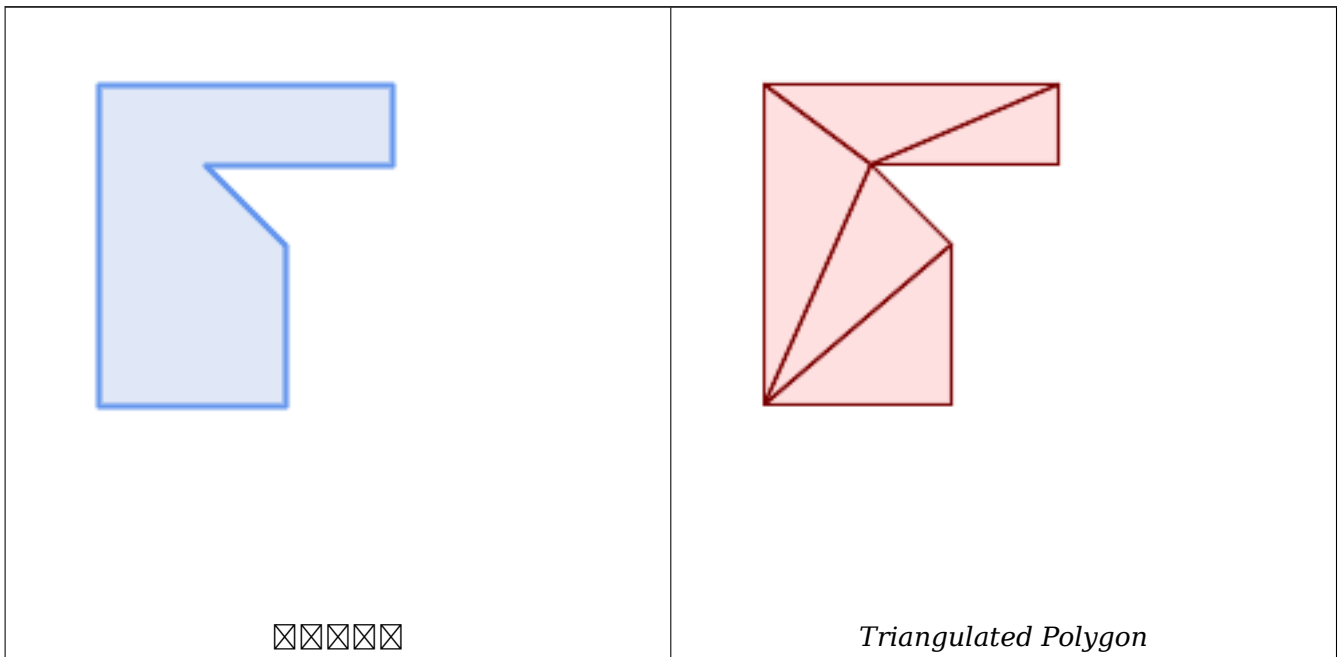
Polygon Triangulation

Same example as ST_Tessellate

```
SELECT ST_TriangulatePolygon(
    'POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160, 120 160, 120 190, 10 190 ←
    ))'::geometry
);
```

ST_AsText 结果:

```
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)))
```



国国

[ST_ConstrainedDelaunayTriangles](#), [ST_DelaunayTriangles](#), [ST_Tessellate](#)

7.14.28 ST_VoronoiLines

ST_VoronoiLines — Returns the boundaries of the Voronoi diagram of the vertices of a geometry.

Synopsis

geometry **ST_VoronoiLines**(geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL);

国国

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.

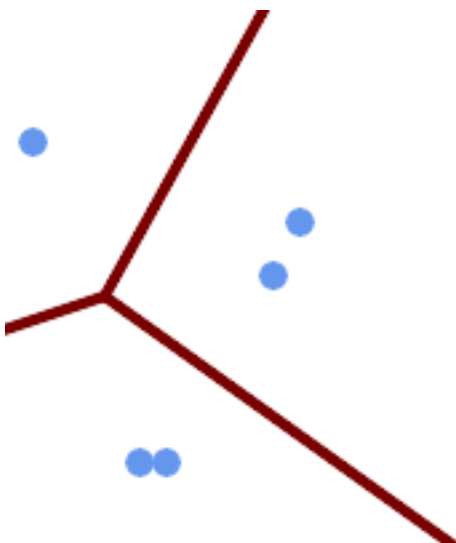
国国国国国国国国国:

- **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 国国国国国

2.3.0 国国国国国国国国国国国国.

国国



Voronoi diagram lines, with tolerance of 30 units

```
SELECT ST_VoronoiLines(
  'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)::geometry,
  30) AS geom;
```

```
ST_AsText output
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273),(36.8181818181818 92.2727272727273,-110 43.3333333333333),(230 -45.7142857142858,36.8181818181818 92.2727272727273))
```

☐☐

[ST_DelaunayTriangles](#), [ST_VoronoiPolygons](#)

7.14.29 ST_VoronoiPolygons

ST_VoronoiPolygons — Returns the cells of the Voronoi diagram of the vertices of a geometry.

Synopsis

geometry **ST_VoronoiPolygons**(geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL);

☐☐

Computes a two-dimensional **Voronoi diagram** from the vertices of the supplied geometry. The result is a GEOMETRYCOLLECTION of POLYGONS that covers an envelope larger than the extent of the input vertices. Returns null if input geometry is null. Returns an empty geometry collection if the input geometry contains only one vertex. Returns an empty geometry collection if the `extend_to` envelope has zero area.

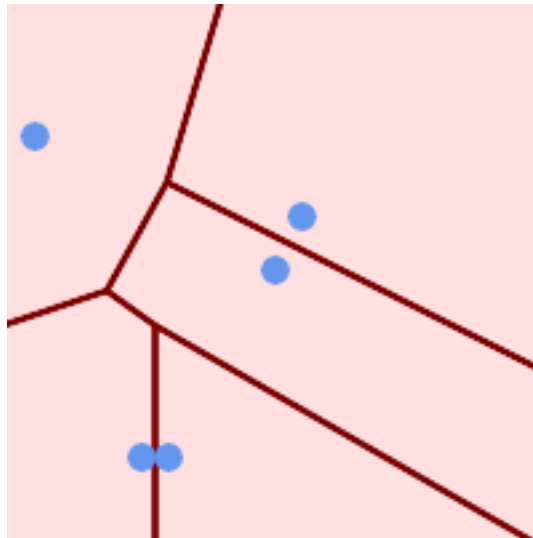
☐☐☐☐☐☐☐☐☐☐:

- **tolerance**: The distance within which vertices will be considered equivalent. Robustness of the algorithm can be improved by supplying a nonzero tolerance distance. (default = 0.0)
- **extend_to**: If present, the diagram is extended to cover the envelope of the supplied geometry, unless smaller than the default envelope (default = NULL, default envelope is the bounding box of the input expanded by about 50%).

GEOS ☐☐☐☐☐

2.3.0 ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐.

图图

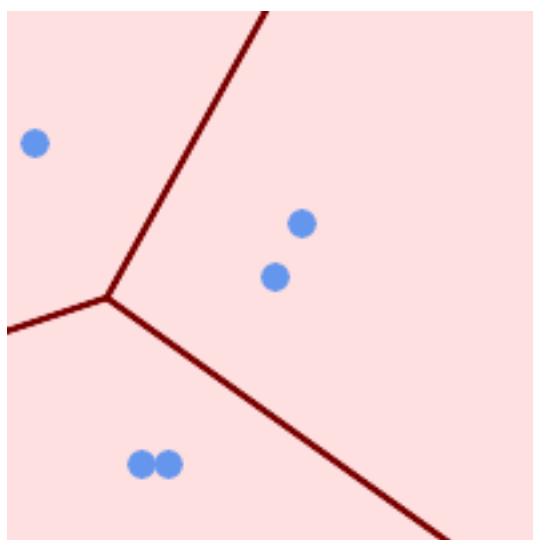


Points overlaid on top of Voronoi diagram

```
SELECT ST_VoronoiPolygons(
    'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120) '::geometry
) AS geom;
```

ST_AsText output

```
GEOMETRYCOLLECTION(POLYGON((-110 43.33333333333333,-110 270,100.5 270,59.3478260869565  ←
    132.826086956522,36.8181818181818 92.2727272727273,-110 43.3333333333333)),
POLYGON((55 -90,-110 -90,-110 43.3333333333333,36.8181818181818 92.2727272727273,55  ←
    79.2857142857143,55 -90)),
POLYGON((230 47.5,230 -20.7142857142857,55 79.2857142857143,36.8181818181818  ←
    92.2727272727273,59.3478260869565 132.826086956522,230 47.5)),POLYGON((230  ←
    -20.7142857142857,230 -90,55 -90,55 79.2857142857143,230 -20.7142857142857)),
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```



Voronoi diagram, with tolerance of 30 units

```
SELECT ST_VoronoiPolygons(
    'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)::geometry,
    30) AS geom;
```

ST_AsText output

```
GEOMETRYCOLLECTION(POLYGON((-110 43.3333333333333,-110 270,100.5 270,59.3478260869565 ↔
    132.826086956522,36.8181818181818 92.2727272727273,-110 43.3333333333333)),
POLYGON((230 47.5,230 -45.7142857142858,36.8181818181818 92.2727272727273,59.3478260869565 ↔
    132.826086956522,230 47.5)),POLYGON((230 -45.7142857142858,230 -90,-110 -90,-110 ↔
    43.3333333333333,36.8181818181818 92.2727272727273,230 -45.7142857142858)),
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```

国国

[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);

国国

A window function which checks if the polygons in the window partition form a valid polygonal coverage. It returns linear indicators showing the location of invalid edges (if any) in each polygon.

A set of valid polygons is a valid coverage if the following conditions hold:

- **Non-overlapping** - polygons do not overlap (their interiors do not intersect)
- **Edge-Matched** - vertices along shared edges are identical

As a window function a value is returned for every input polygon. For polygons which violate one or more of the validity conditions the return value is a MULTILINESTRING containing the problematic edges. Coverage-valid polygons return the value NULL. Non-polygonal or empty geometries also produce NULL values.

The conditions allow a valid coverage to contain holes (gaps between polygons), as long as the surrounding polygons are edge-matched. However, very narrow gaps are often undesirable. If the *tolerance* parameter is specified with a non-zero distance, edges forming narrower gaps will also be returned as invalid.

The polygons being checked for coverage validity must also be valid geometries. This can be checked with [ST_IsValid](#).

Availability: 3.4.0

Requires GEOS >= 3.12.0



Invalid edges caused by overlap and non-matching vertices

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 190, 30 160, 40 110, 100 70, 120 10, 10 10, 10 190))'::geometry),
  (2, 'POLYGON ((100 190, 10 190, 30 160, 40 110, 50 80, 74 110.5, 100 130, 140 120, 140 160, 100 190))'::geometry),
  (3, 'POLYGON ((140 190, 190 190, 190 80, 140 80, 140 190))'::geometry),
  (4, 'POLYGON ((180 40, 120 10, 100 70, 140 80, 190 80, 180 40))'::geometry)
)
SELECT id, ST_AsText(ST_CoverageInvalidEdges(geom) OVER ())
FROM coverage;
```

id	st_astext
1	LINESTRING (40 110, 100 70)
2	MULTILINESTRING ((100 130, 140 120, 140 160, 100 190), (40 110, 50 80, 74 110.5))
3	LINESTRING (140 80, 140 190)
4	null

```
-- Test entire table for coverage validity
SELECT true = ALL (
  SELECT ST_CoverageInvalidEdges(geom) OVER () IS NULL
  FROM coverage
);
```



[ST_IsValid](#), [ST_CoverageUnion](#), [ST_CoverageClean](#), [ST_CoverageSimplify](#)

7.15.2 ST_CoverageSimplify

`ST_CoverageSimplify` — Window function that simplifies the edges of a polygonal coverage.

Synopsis

geometry **ST_CoverageSimplify**(geometry winset geom, float8 tolerance, boolean simplifyBoundary = true);

A window function which simplifies the edges of polygons in a polygonal coverage. The simplification preserves the coverage topology. This means the simplified output polygons are consistent along shared edges, and still form a valid coverage.

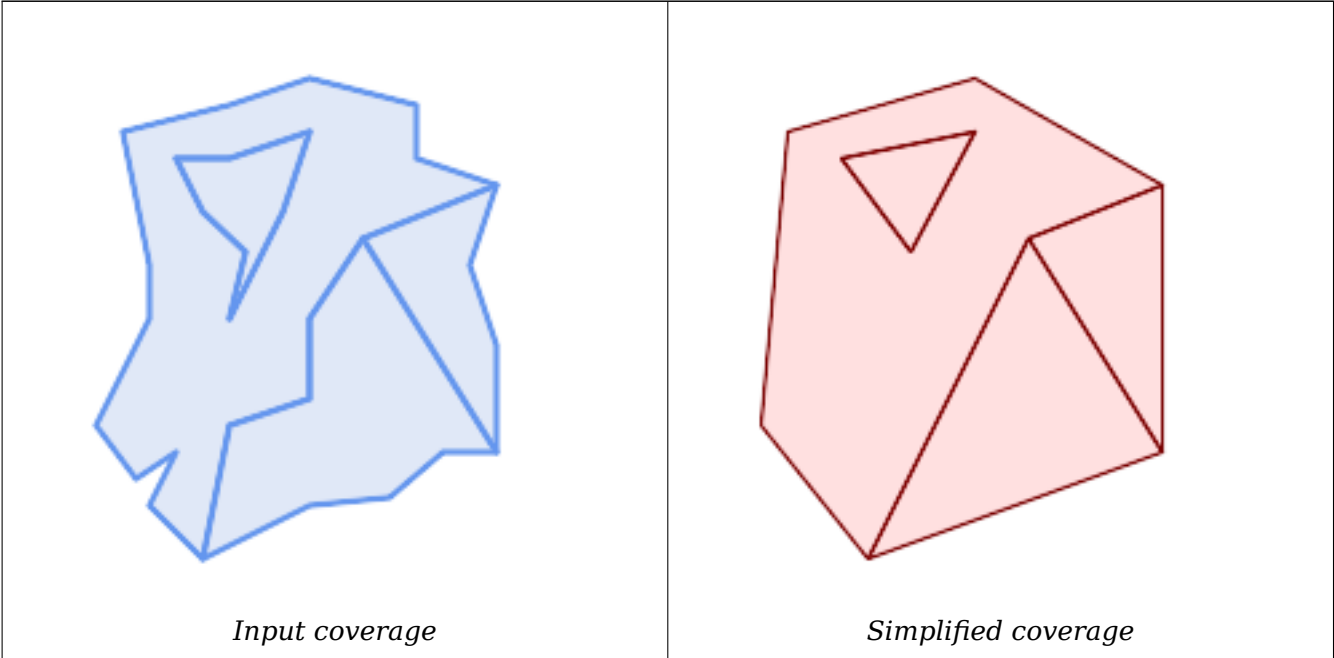
The simplification uses a variant of the **Visvalingam-Whyatt algorithm**. The *tolerance* parameter has units of distance, and is roughly equal to the square root of triangular areas to be simplified.

To simplify only the "internal" edges of the coverage (those that are shared by two polygons) set the *simplifyBoundary* parameter to false.

Note

If the input is not a valid coverage there may be unexpected artifacts in the output (such as boundary intersections, or separated boundaries which appeared to be shared). Use **ST_CoverageInvalidEdges** to determine if a coverage is valid.

Availability: 3.4.0
Requires GEOS >= 3.12.0



```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((160 150, 110 130, 90 100, 90 70, 60 60, 50 10, 30 30, 40 50, 25 40, 10 60, 30 100, 30 120, 20 170, 60 180, 90 190, 130 180, 130 160, 160 150), (40 160, 50 140, 66 125, 60 100, 80 140, 90 170, 60 160, 40 160))'::geometry),
```

```
(2, 'POLYGON ((40 160, 60 160, 90 170, 80 140, 60 100, 66 125, 50 140, 40 160))':: geometry),
(3, 'POLYGON ((110 130, 160 50, 140 50, 120 33, 90 30, 50 10, 60 60, 90 70, 90 100, 110 130))'::geometry),
(4, 'POLYGON ((160 150, 150 120, 160 90, 160 50, 110 130, 160 150))'::geometry)
)
SELECT id, ST_AsText(ST_CoverageSimplify(geom, 30) OVER ())
FROM coverage;
```

id	st_astext
1	POLYGON ((160 150, 110 130, 50 10, 10 60, 20 170, 90 190, 160 150), (40 160, 66 125, 90 170, 40 160))
2	POLYGON ((40 160, 66 125, 90 170, 40 160))
3	POLYGON ((110 130, 160 50, 50 10, 110 130))
4	POLYGON ((160 150, 160 50, 110 130, 160 150))



[ST_CoverageInvalidEdges](#), [ST_CoverageUnion](#), [ST_CoverageClean](#)

7.15.3 ST_CoverageUnion

ST_CoverageUnion — Computes the union of a set of polygons forming a coverage by removing shared edges.

Synopsis

geometry **ST_CoverageUnion**(geometry set geom);



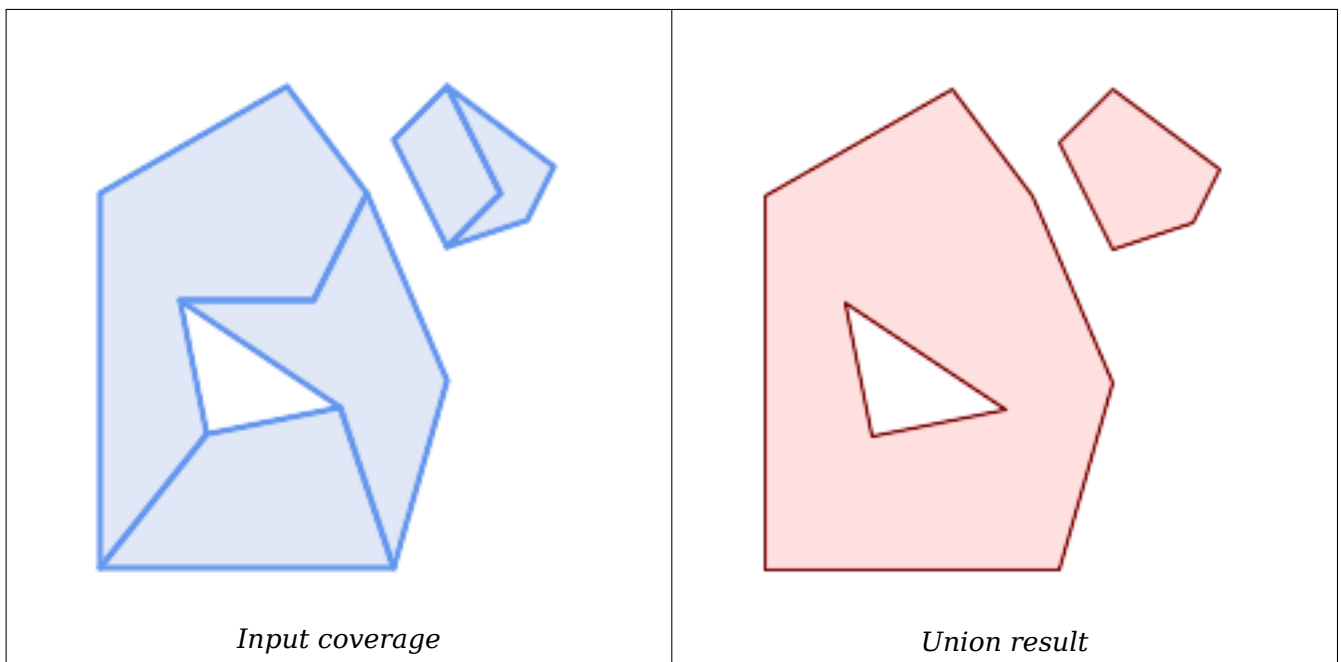
An aggregate function which unions a set of polygons forming a polygonal coverage. The result is a polygonal geometry covering the same area as the coverage. This function produces the same result as [ST_Union](#), but uses the coverage structure to compute the union much faster.



Note If the input is not a valid coverage there may be unexpected artifacts in the output (such as unmerged or overlapping polygons). Use [ST_CoverageInvalidEdges](#) to determine if a coverage is valid.

Availability: 3.4.0 - requires GEOS >= 3.8.0





```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 10, 10 150, 80 190, 110 150, 90 110, 40 110, 50 60, 10 10))'::geometry) ←
  ,
  (2, 'POLYGON ((120 10, 10 10, 50 60, 100 70, 120 10))'::geometry),
  (3, 'POLYGON ((140 80, 120 10, 100 70, 40 110, 90 110, 110 150, 140 80))'::geometry),
  (4, 'POLYGON ((140 190, 120 170, 140 130, 160 150, 140 190))'::geometry),
  (5, 'POLYGON ((180 160, 170 140, 140 130, 160 150, 140 190, 180 160))'::geometry)
)
SELECT ST_AsText(ST_CoverageUnion(geom))
FROM coverage;

-----
MULTIPOLYGON (((10 150, 80 190, 110 150, 140 80, 120 10, 10 10, 10 150), (50 60, 100 70, 40 ←
  110, 50 60)), ((120 170, 140 190, 180 160, 170 140, 140 130, 120 170)))
```

国国

[ST_CoverageInvalidEdges](#), [ST_CoverageSimplify](#), [ST_CoverageClean](#), [ST_Union](#)

7.15.4 ST_CoverageClean

ST_CoverageClean — Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.

Synopsis

geometry **ST_CoverageClean**(geometry winset geom, float8 gapMaximumWidth = 0, float8 snappingDistance = -1, text overlapMergeStrategy = 'MERGE_LONGEST_BORDER');

国国

A window function which alters the edges of a polygonal coverage to ensure that none of the polygons overlap, that small gaps are snapped away, and that all shared edges are exactly identical. The result is a clean coverage that will pass validation tests like [ST_CoverageInvalidEdges](#)

The *gapMaximumWidth* controls the cleaning of gaps between polygons. Gaps smaller than this tolerance will be closed.

The *snappingDistance* controls the node snapping step, when nearby vertices are snapped together. The default setting (-1) applies an automatic snapping distance based on an analysis of the input. Set to 0.0 to turn off all snapping.

The *overlapMergeStrategy* controls the algorithm used to determine which neighboring polygons to merge overlapping areas into.

MERGE_LONGEST_BORDER chooses polygon with longest common border

MERGE_MAX_AREA chooses polygon with maximum area

MERGE_MIN_AREA chooses polygon with minimum area

MERGE_MIN_INDEX chooses polygon with smallest input index

Availability: 3.6.0 - requires GEOS >= 3.14.0

国国

```
-- Populate demo table
CREATE TABLE example AS SELECT * FROM (VALUES
  (1, 'POLYGON ((10 190, 30 160, 40 110, 100 70, 120 10, 10 10, 10 190))'::geometry),
  (2, 'POLYGON ((100 190, 10 190, 30 160, 40 110, 50 80, 74 110.5, 100 130, 140 120, 140 160, 100 190))'::geometry),
  (3, 'POLYGON ((140 190, 190 190, 190 80, 140 80, 140 190))'::geometry),
  (4, 'POLYGON ((180 40, 120 10, 100 70, 140 80, 190 80, 180 40))'::geometry)
) AS v(id, geom);

-- Prove it is a dirty coverage
SELECT ST_AsText(ST_CoverageInvalidEdges(geom) OVER ())
FROM example;

-- Clean the coverage
CREATE TABLE example_clean AS
  SELECT id, ST_CoverageClean(geom) OVER () AS GEOM
  FROM example;

-- Prove it is a clean coverage
SELECT ST_AsText(ST_CoverageInvalidEdges(geom) OVER ())
FROM example_clean;
```

国国

[ST_CoverageInvalidEdges](#), [ST_Union](#) [ST_CoverageSimplify](#)

7.16 Affine Transformations

7.16.1 ST_Affine

ST_Affine — Apply a 3D affine transformation to a geometry.

Synopsis

geometry **ST_Affine**(geometry geomA, float a, float b, float c, float d, float e, float f, float g, float h, float i, float xoff, float yoff, float zoff);
 geometry **ST_Affine**(geometry geomA, float a, float b, float d, float e, float xoff, float yoff);

Notes

Applies a 3D affine transformation to the geometry to do things like translate, rotate, scale in one step.

Version 1: The call

```
ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)
```

represents the transformation matrix

```
/ a b c xoff \
| d e f yoff |
| g h i zoff |
\ 0 0 0 1 /
```

and the vertices are transformed as follows:

```
x' = a*x + b*y + c*z + xoff
y' = d*x + e*y + f*z + yoff
z' = g*x + h*y + i*z + zoff
```

All of the translate / scale functions below are expressed via such an affine transformation.

Version 2: Applies a 2d affine transformation to the geometry. The call

```
ST_Affine(geom, a, b, d, e, xoff, yoff)
```

represents the transformation matrix

```
/ a b 0 xoff \      / a b xoff \
| d e 0 yoff |  rsp. | d e yoff |
| 0 0 1 0  |      \ 0 0 1 /
\ 0 0 0 1 /
```

and the vertices are transformed as follows:

```
x' = a*x + b*y + xoff
y' = d*x + e*y + yoff
z' = z
```

This method is a subcase of the 3D method above.

History: 2.0.0 **Added**, **Added** TIN **Added**.

Availability: 1.1.2. Name changed from Affine to ST_Affine in 1.2.2



Note

1.3.4 **Added** (curve) **Added**. 1.3.4 **Added**.

- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.

国国

```
--Rotate a 3d line 180 degrees about the z axis. Note this is long-hand for doing ↵
ST_Rotate();
SELECT ST_AseWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), 0, 0, ↵
    0, 1, 0, 0, 0)) As using_affine,
    ST_AseWKT(ST_Rotate(geom, pi())) As using_rotate
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
      using_affine      |      using_rotate
-----+-----
LINESTRING(-1 -2 3,-1 -4 3) | LINESTRING(-1 -2 3,-1 -4 3)
(1 row)

--Rotate a 3d line 180 degrees in both the x and z axis
SELECT ST_AseWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin(pi()) ↵
    , 0, sin(pi()), cos(pi()), 0, 0, 0))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
      st_asewkt
-----
LINESTRING(-1 -2 -3,-1 -4 -3)
(1 row)
```

国国

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);
```

国国

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).

国国国国: 2.0.0 国国国国国国国国, 国国国国 TIN 国国国国国国国国国国.

Enhanced: 2.0.0 additional parameters for specifying the origin of rotation were added.

Availability: 1.1.2. Name changed from Rotate to ST_Rotate in 1.2.2

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

￼

```
--Rotate 180 degrees
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()));
           st_asewkt
-----
LINESTRING(-50 -160,-50 -50,-100 -50)
(1 row)

--Rotate 30 degrees counter-clockwise at x=50, y=160
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()/6, 50, 160));
           st_asewkt
-----
LINESTRING(50 160,105 64.7372055837117,148.301270189222 89.7372055837117)
(1 row)

--Rotate 60 degrees clockwise from centroid
SELECT ST_AsEWKT(ST_Rotate(geom, -pi()/3, ST_Centroid(geom)))
FROM (SELECT 'LINESTRING (50 160, 50 50, 100 50) '::geometry AS geom) AS foo;
           st_asewkt
-----
LINESTRING(116.4225 130.6721,21.1597 75.6721,46.1597 32.3708)
(1 row)
```

￼

ST_Affine, ST_RotateX, ST_RotateY, ST_RotateZ

7.16.3 ST_RotateX

ST_RotateX — Rotates a geometry about the X axis.

Synopsis

geometry **ST_RotateX**(geometry geomA, float rotRadians);

￼

Rotates a geometry geomA - rotRadians about the X axis.

**Note**

ST_RotateX(geomA, rotRadians) is short-hand for ST_Affine(geomA, 1, 0, 0, 0, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0).

变更: 2.0.0 支持多面体, 支持 TIN 表面。

Availability: 1.1.2. Name changed from RotateX to ST_RotateX in 1.2.2



This function supports Polyhedral surfaces.



This function supports 3d and will not drop the z-index.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

例

```
--Rotate a line 90 degrees along x-axis
SELECT ST_AsEWKT(ST_RotateX(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
           st_asewkt
-----
LINESTRING(1 -3 2,1 -1 1)
```

例

[ST_Affine](#), [ST_RotateY](#), [ST_RotateZ](#)

7.16.4 ST_RotateY

ST_RotateY — Rotates a geometry about the Y axis.

Synopsis

geometry **ST_RotateY**(geometry geomA, float rotRadians);

例

Rotates a geometry geomA - rotRadians about the y axis.

**Note**

ST_RotateY(geomA, rotRadians) is short-hand for ST_Affine(geomA, cos(rotRadians), 0, sin(rotRadians), 0, 1, 0, -sin(rotRadians), 0, cos(rotRadians), 0, 0, 0).

Availability: 1.1.2. Name changed from RotateY to ST_RotateY in 1.2.2

变更: 2.0.0 支持多面体, 支持 TIN 表面。



This function supports Polyhedral surfaces.



This function supports 3d and will not drop the z-index.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

例

```
--Rotate a line 90 degrees along y-axis
SELECT ST_AsEWKT(ST_RotateY(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
           st_asewkt
-----
LINESTRING(3 2 -1,1 1 -1)
```

例

ST_Affine, ST_RotateX, ST_RotateZ

7.16.5 ST_RotateZ

ST_RotateZ — Rotates a geometry about the Z axis.

Synopsis

geometry **ST_RotateZ**(geometry geomA, float rotRadians);

例

Rotates a geometry geomA - rotRadians about the Z axis.



Note

This is a synonym for ST_Rotate



Note

ST_RotateZ(geomA, rotRadians) is short-hand for SELECT ST_Affine(geomA, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0, 1, 0, 0, 0).

更新: 2.0.0 版本, 支持 TIN 数据类型。

Availability: 1.1.2. Name changed from RotateZ to ST_RotateZ in 1.2.2



Note

1.3.4 版本 (curve) 支持。 1.3.4 版本支持。



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).

❏

```
--Rotate a line 90 degrees along z-axis
SELECT ST_AsEWKT(ST_RotateZ(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
           st_asewkt
-----
LINESTRING(-2 1 3,-1 1 1)

--Rotate a curved circle around z-axis
SELECT ST_AsEWKT(ST_RotateZ(geom, pi()/2))
FROM (SELECT ST_LineToCurve(ST_Buffer(ST_GeomFromText('POINT(234 567)'), 3)) As geom) As  ↵
      foo;

-----

CURVEPOLYGON(CIRCULARSTRING(-567 237,-564.87867965644 236.12132034356,-564  ↵
234,-569.12132034356 231.87867965644,-567 237))
```

❏

ST_Affine, ST_RotateX, ST_RotateY

7.16.6 ST_Scale

ST_Scale — Scales a geometry by given factors.

Synopsis

```
geometry ST_Scale(geometry geomA, float XFactor, float YFactor, float ZFactor);
geometry ST_Scale(geometry geomA, float XFactor, float YFactor);
geometry ST_Scale(geometry geom, geometry factor);
geometry ST_Scale(geometry geom, geometry factor, geometry origin);
```

❏

Scales the geometry to a new size by multiplying the ordinates with the corresponding factor parameters.

The version taking a geometry as the **factor** parameter allows passing a 2d, 3dm, 3dz or 4d point to set scaling factor for all supported dimensions. Missing dimensions in the **factor** point are equivalent to no scaling the corresponding dimension.

The three-geometry variant allows a “false origin” for the scaling to be passed in. This allows “scaling in place”, for example using the centroid of the geometry as the false origin. Without a false origin, scaling takes place relative to the actual origin, so all coordinates are just multiplied by the scale factor.



Note

1.3.4 简体中文 (curve) 简体中文. 1.3.4 简体中文.

Availability: 1.1.0.

版本: 2.0.0 引入, 支持 TIN 表面。

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.

SQL

```
--Version 1: scale X, Y, Z
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75, 0.8));
           st_asewkt
-----
LINESTRING(0.5 1.5 2.4,0.5 0.75 0.8)

--Version 2: Scale X Y
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75));
           st_asewkt
-----
LINESTRING(0.5 1.5 3,0.5 0.75 1)

--Version 3: Scale X Y Z M
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)'),
  ST_MakePoint(0.5, 0.75, 2, -1)));
           st_asewkt
-----
LINESTRING(0.5 1.5 6 -4,0.5 0.75 2 -1)

--Version 4: Scale X Y using false origin
SELECT ST_AsText(ST_Scale('LINESTRING(1 1, 2 2)', 'POINT(2 2)', 'POINT(1 1)::geometry'));
           st_astext
-----
LINESTRING(1 1,3 3)
```

SQL

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);

返回

Returns a new geometry whose coordinates are translated delta x,delta y,delta z units. Units are based on the units defined in spatial reference (SRID) for this geometry.



Note

1.3.4 版本中，**ST_Translate** 支持 (curve) 类型。1.3.4 版本中，**ST_Translate** 支持 Circular Strings 和 Curves。

1.2.2 版本中，**ST_Translate** 支持。



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

返回

Move a point 1 degree longitude

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('POINT(-71.01 42.37)',4326),1,0)) As wgs_transgeomtxt;

wgs_transgeomtxt
-----
POINT(-70.01 42.37)
```

Move a linestring 1 degree longitude and 1/2 degree latitude

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('LINESTRING(-71.01 42.37,-71.11 42.38)',4326),1,0.5)) As wgs_transgeomtxt;

wgs_transgeomtxt
-----
LINESTRING(-70.01 42.87,-70.11 42.88)
```

Move a 3d point

```
SELECT ST_AsEWKT(ST_Translate(CAST('POINT(0 0 0)' As geometry), 5, 12,3));

st_asewkt
-----
POINT(5 12 3)
```

Move a curve and a point

```
SELECT ST_AsText(ST_Translate(ST_Collect('CURVEPOLYGON(CIRCULARSTRING(4 3,3.12 0.878,1 0,-1.121 5.1213,6 7, 8 9,4 3))','POINT(1 3)'),1,2));
```

```
-----
GEOMETRYCOLLECTION(CURVEPOLYGON(CIRCULARSTRING(5 5,4.12 2.878,2 2,-0.121 7.1213,7 9,9 11,5 5)),POINT(2 5))
```

函数

[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);

函数

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 版本中，`ST_TransScale` 对曲线（curve）的支持是不完整的。1.3.4 版本中，`ST_TransScale` 对曲线（curve）的支持是不完整的。

Availability: 1.1.0.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

函数

```
SELECT ST_AsEWKT(ST_TransScale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 1, 1, 2));
           st_asewkt
-----
LINESTRING(1.5 6 3,1.5 4 1)
```

```
--Buffer a point to get an approximation of a circle, convert to curve and then translate ↩
1,2 and scale it 3,4
SELECT ST_AsText(ST_Transscale(ST_LineToCurve(ST_Buffer('POINT(234 567)', 3)),1,2,3,4));
```

```
-----
CURVEPOLYGON(CIRCULARSTRING(714 2276,711.363961030679 2267.51471862576,705 ↩
2264,698.636038969321 2284.48528137424,714 2276))
```

☒☒

[ST_Affine](#), [ST_Translate](#)

7.17 Clustering Functions

7.17.1 ST_ClusterDBSCAN

`ST_ClusterDBSCAN` — Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.

Synopsis

integer **ST_ClusterDBSCAN**(geometry winset geom, float8 eps, integer minpoints);

☒☒

A window function that returns a cluster number for each input geometry, using the 2D [Density-based spatial clustering of applications with noise \(DBSCAN\)](#) algorithm. Unlike [ST_ClusterKMeans](#), it does not require the number of clusters to be specified, but instead uses the desired [distance](#) (eps) and density (minpoints) parameters to determine each cluster.

An input geometry is added to a cluster if it is either:

- A "core" geometry, that is within eps [distance](#) of at least minpoints input geometries (including itself); or
- A "border" geometry, that is within eps [distance](#) of a core geometry.

Note that border geometries may be within eps distance of core geometries in more than one cluster. Either assignment would be correct, so the border geometry will be arbitrarily assigned to one of the available clusters. In this situation it is possible for a correct cluster to be generated with fewer than minpoints geometries. To ensure deterministic assignment of border geometries (so that repeated calls to `ST_ClusterDBSCAN` will produce identical results) use an `ORDER BY` clause in the window definition. Ambiguous cluster assignments may differ from other DBSCAN implementations.



Note

Geometries that do not meet the criteria to join any cluster are assigned a cluster number of NULL.

2.3.0 ☒☒☒☒☒☒☒☒☒☒.



This method supports Circular Strings and Curves.

☒☒

Clustering polygon within 50 meters of each other, and requiring at least 2 polygons per cluster.

Clusters within 50 meters with at least 2 items per cluster. Singletons have NULL for cid

```
SELECT name, ST_ClusterDBSCAN(geom, eps = > 50, minpoints = > 2) over () AS cid
FROM boston_polys
WHERE name
> '' AND building
> ''
AND ST_DWithin(geom,
ST_Transform(
ST_GeomFromText('POINT ↵
(-71.04054 42.35141)', 4326), 26986),
500);
```

bucket	name		↵
Manulife Tower			↵
0			
Park Lane Seaport I			↵
0			
Park Lane Seaport II			↵
0			
Renaissance Boston Waterfront Hotel			↵
0			
Seaport Boston Hotel			↵
0			
Seaport Hotel & World Trade Center			↵
0			
Waterside Place			↵
0			
World Trade Center East			↵
0			
100 Northern Avenue			↵
1			
100 Pier 4			↵
1			
The Institute of Contemporary Art			↵
1			
101 Seaport			↵
2			
District Hall			↵
2			
One Marina Park Drive			↵
2			
Twenty Two Liberty			↵
2			
Vertex			↵
2			
Vertex			↵
2			
Watermark Seaport			↵
2			
Blue Hills Bank Pavilion			↵
NULL			
World Trade Center West			↵
NULL			
(20 rows)			

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);

国国

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 国国国国国国国国国国.

国国

```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)::geometry',
    'LINESTRING (5 5, 4 4)::geometry',
    'LINESTRING (6 6, 7 7)::geometry',
    'LINESTRING (0 0, -1 -1)::geometry',
    'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))::geometry']) AS geom)

SELECT ST_AsText(unnest(ST_ClusterIntersecting(geom))) FROM testdata;

-- result

st_astext
-----
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

国国

[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);

描述

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

用法

```
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

相关链接

[ST_ClusterIntersecting](#), [ST_ClusterWithin](#), [ST_ClusterWithinWin](#)

7.17.4 ST_ClusterKMeans

ST_ClusterKMeans — Window function that returns a cluster id for each input geometry using the K-means algorithm.

Synopsis

integer **ST_ClusterKMeans**(geometry winset geom , integer k , float8 max_radius);

描述

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

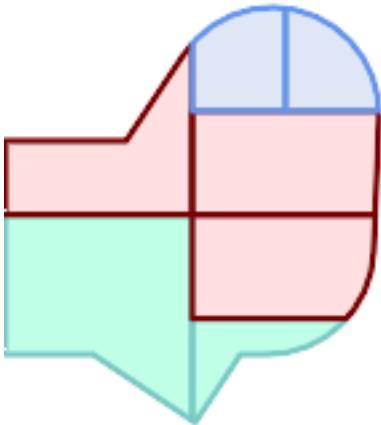
Enhanced: 3.1.0 Support for 3D geometries and weights

2.3.0 支持多边形。

¶

Generate dummy set of parcels for examples:

```
CREATE TABLE parcels AS
SELECT lpad((row_number() over())::text,3,'0') As parcel_id, geom,
('{residential, commercial}'::text[])[1 + mod(row_number()OVER(),2)] As type
FROM
  ST_Subdivide(ST_Buffer('SRID=3857;LINESTRING(40 100, 98 100, 100 150, 60 90)'::geometry ←
    40, 'endcap=square'),12) As geom;
```



Parcels color-coded by cluster number (cid)

```
SELECT ST_ClusterKMeans(geom, 3) OVER() AS cid, parcel_id, geom
FROM parcels;
```

cid	parcel_id	geom
0	001	0103000000...
0	002	0103000000...
1	003	0103000000...
0	004	0103000000...
1	005	0103000000...
2	006	0103000000...
2	007	0103000000...

Partitioning parcel clusters by type:

```
SELECT ST_ClusterKMeans(geom, 3) over (PARTITION BY type) AS cid, parcel_id, type
FROM parcels;
```

cid	parcel_id	type
1	005	commercial
1	003	commercial
2	007	commercial
0	001	commercial
1	004	residential
0	002	residential
2	006	residential

Example: Clustering a preaggregated planetary-scale data population dataset using 3D clustering and weighting. Identify at least 20 regions based on **Kontur Population Data** that do not span more than 3000 km from their center:

```
create table kontur_population_3000km_clusters as
select
  geom,
  ST_ClusterKMeans(
    ST_Force4D(
      ST_Transform(ST_Force3D(geom), 4978), -- cluster in 3D XYZ CRS
      mvalue => population -- set clustering to be weighed by population
    ),
    20, -- aim to generate at least 20 clusters
    max_radius => 3000000 -- but generate more to make each under 3000 km radius
  ) over () as cid
from
  kontur_population;
```



World population clustered to above specs produces 46 clusters. Clusters are centered at well-populated regions (New York, Moscow). Greenland is one cluster. There are island clusters that span across the antimeridian. Cluster edges follow Earth's curvature.

￼

ST_ClusterDBSCAN, **ST_ClusterIntersectingWin**, **ST_ClusterWithinWin**, **ST_ClusterIntersecting**, **ST_ClusterST_Subdivide**, **ST_Force3D**, **ST_Force4D**,

7.17.5 ST_ClusterWithin

ST_ClusterWithin — Aggregate function that clusters geometries by separation distance.

Synopsis

```
geometry[] ST_ClusterWithin(geometry set g, float8 distance);
```



An aggregate function that returns an array of GeometryCollections, where each collection is a cluster containing some input geometries. Clustering partitions the input geometries into sets in which each geometry is within the specified *distance* of at least one other geometry in the same cluster. Distances are Cartesian distances in the units of the SRID.

ST_ClusterWithin is equivalent to running **ST_ClusterDBSCAN** with minpoints => 0.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.



This method supports Circular Strings and Curves.



```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1) '::geometry,
                      'LINESTRING (5 5, 4 4) '::geometry,
                      'LINESTRING (6 6, 7 7) '::geometry,
                      'LINESTRING (0 0, -1 -1) '::geometry,
                      'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))' '::geometry]) AS geom)

SELECT ST_AsText(unnest(ST_ClusterWithin(geom, 1.4))) FROM testdata;

--result

st_astext
-----
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```



ST ClusterWithinWin, ST ClusterDBSCAN, ST ClusterIntersecting, ST ClusterIntersectingWin

7.17.6 ST ClusterWithinWin

ST_ClusterWithinWin — Window function that returns a cluster id for each input geometry, clustering using separation distance.

Synopsis

```
integer ST_ClusterWithinWin(geometry winset geom, float8 distance);
```



A window function that returns a cluster number for each input geometry. Clustering partitions the geometries into sets in which each geometry is within the specified distance of at least one other geometry in the same cluster. Distances are Cartesian distances in the units of the SRID.

ST ClusterWithinWin is equivalent to running **ST ClusterDBSCAN** with minpoints $\Rightarrow 0$.

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_ClusterWithin(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)'));
```

```
box2d
```

```
-----
```

```
BOX(1 2,5 6)
```

```
SELECT Box2D(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'));
```

```
box2d
```

```
-----
```

```
BOX(220186.984375 150406,220288.25 150506.140625)
```

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 几何体。

- ✓ 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
```

```
-----
```

```
BOX3D(1 2 3,5 6 5)
```

```
SELECT Box3D(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 1,220227 150406 1)'));
```

```
Box3d
```

```
-----
```

```
BOX3D(220227 150406 1,220268 150415 1)
```

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.

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');
--result--
BOX(-8877653 4912316,-8010225.5 5589284)

SELECT ST_EstimatedExtent('feature_poly', 'geom');
--result--
BOX(-124.659652709961 24.6830825805664,-67.7798080444336 49.0012092590332)
```

新新

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);
```

新新

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.

新新新: 2.0.0 新新新新新新新, 新新新 TIN 新新新新新新新新新.

Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒



Note

Examples below use US National Atlas Equal Area (SRID=2163) which is a meter projection

```
--10 meter expanded box around bbox of a linestring
SELECT CAST(ST_Expand(ST_GeomFromText('LINESTRING(2312980 110676,2312923 110701,2312892 110714)', 2163),10) As box2d);
                                st_expand
-----
BOX(2312882 110666,2312990 110724)

--10 meter expanded 3D box of a 3D box
SELECT ST_Expand(CAST('BOX3D(778783 2951741 1,794875 2970042.61545891 10)' As box3d),10)
                                st_expand
-----
BOX3D(778773 2951731 -9,794885 2970052.61545891 20)

--10 meter geometry astext rep of a expand box around a point geometry
SELECT ST_AsEWKT(ST_Expand(ST_GeomFromEWKT('SRID=2163;POINT(2312980 110676)'),10));
                                st_asewkt
-----
SRID=2163;POLYGON((2312970 110666,2312970 110686,2312990 110686,2312990 110666,2312970 110666))
```

☒☒

[ST_Buffer](#), [ST_DWithin](#), [ST_SRID](#)

7.18.5 ST_Extent

ST_Extent — Aggregate function that returns the bounding box of geometries.

Synopsis

box2d **ST_Extent**(geometry set geomfield);

☒☒

An aggregate function that returns a **box2d** bounding box that bounds a set of geometries. The bounding box coordinates are in the spatial reference system of the input geometries. **ST_Extent** is similar in concept to Oracle Spatial/Locator's **SDO_AGGR_MBR**.



Note

ST_Extent returns boxes with only X and Y ordinates even with 3D geometries. To return XYZ ordinates use **ST_3DExtent**.



Note
The returned box3d value does not include a SRID. Use `ST_SetSRID` to convert it into a geometry with SRID metadata. The SRID is the same as the input geometries.

PostGIS: 2.0.0 简体中文, 支持 TIN 数据类型。

- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☐☐



Note
Examples below use Massachusetts State Plane ft (SRID=2249)

```
SELECT ST_Extent(geom) as bextent FROM sometable;
                                st_bextent
-----
BOX(739651.875 2908247.25,794875.8125 2970042.75)

--Return extent of each category of geometries
SELECT ST_Extent(geom) as bextent
FROM sometable
GROUP BY category ORDER BY category;

                                bextent                                |          name
-----+-----
BOX(778783.5625 2951741.25,794875.8125 2970042.75) | A
BOX(751315.8125 2919164.75,765202.6875 2935417.25) | B
BOX(739651.875 2917394.75,756688.375 2935866)      | C

--Force back into a geometry
-- and render the extended text representation of that geometry
SELECT ST_SetSRID(ST_Extent(geom),2249) as bextent FROM sometable;

                                bextent
-----
SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,
794875.8125 2908247.25,739651.875 2908247.25))
```

☐☐

`ST_EstimatedExtent`, `ST_3DExtent`, `ST_SetSRID`

7.18.6 ST_3DExtent

`ST_3DExtent` — Aggregate function that returns the 3D bounding box of geometries.

Synopsis

box3d **ST_3DExtent**(geometry set geomfield);

⚠

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 弃用, 使用 **TIN** 代替。

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;
      b3extent
-----
BOX3D(1 1 0,3 2 2)

--Get the extent of various elevated circular strings
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_Translate(ST_Force_3DZ(ST_LineToCurve(ST_Buffer(ST_Point(x,y),1))),0,0,z)
      As geom
      FROM generate_series(1,3) As x
      CROSS JOIN generate_series(1,2) As y
      CROSS JOIN generate_series(0,2) As Z) As foo;
      b3extent
-----
BOX3D(1 0 0,4 2 2)
```

⚠

ST_Extent, **ST_Force3DZ**, **ST_SetSRID**

7.18.7 ST_MakeBox2D

ST_MakeBox2D — Creates a BOX2D defined by two 2D point geometries.

Synopsis

box2d **ST_MakeBox2D**(geometry pointLowLeft, geometry pointUpRight);



Creates a **box2d** defined by two Point geometries. This is useful for doing range queries.



```
--Return all features that fall reside or partly reside in a US national atlas coordinate ↵
    bounding box
--It is assumed here that the geometries are stored with SRID = 2163 (US National atlas ↵
    equal area)
SELECT feature_id, feature_name, geom
FROM features
WHERE geom && ST_SetSRID(ST_MakeBox2D(ST_Point(-989502.1875, 528439.5625),
    ST_Point(-987121.375 ,529933.1875)),2163)
```



ST_Point, **ST_SetSRID**, **ST_SRID**

7.18.8 ST_3DMakeBox

ST_3DMakeBox — Creates a BOX3D defined by two 3D point geometries.

Synopsis

box3d **ST_3DMakeBox**(geometry point3DLowLeftBottom, geometry point3DUpRightTop);



Creates a **box3d** defined by two 3D Point geometries.



This function supports 3D and will not drop the z-index.

Changed: 2.0.0 In prior versions this used to be called ST_MakeBox3D



```
SELECT ST_3DMakeBox(ST_MakePoint(-989502.1875, 528439.5625, 10),
    ST_MakePoint(-987121.375 ,529933.1875, 10)) As abb3d
--bb3d--
-----
BOX3D(-989502.1875 528439.5625 10,-987121.375 529933.1875 10)
```

☒☒

[ST_MakePoint](#), [ST_SetSRID](#), [ST_SRID](#)

7.18.9 ST_XMax

ST_XMax — Returns the X maxima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_XMax**(box3d aGeomorBox2DorBox3D);

☒☒

Returns the X maxima of a 2D or 3D bounding box or a geometry.



Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However, it will not accept a geometry or box2d text representation, since those do not auto-cast.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_XMax('BOX3D(1 2 3, 4 5 6)');
st_xmax
-----
4

SELECT ST_XMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmax
-----
5

SELECT ST_XMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmax
-----
3
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
a BOX3D
SELECT ST_XMax('LINESTRING(1 3, 5 6)');
--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
150406 3)'));
st_xmax
-----
220288.248780547
```

☒☒

`ST_XMin`, `ST_YMax`, `ST_YMin`, `ST_ZMax`, `ST_ZMin`

7.18.10 ST_XMin

`ST_XMin` — Returns the X minima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_XMin**(box3d aGeomorBox2DorBox3D);

☒☒

Returns the X minima of a 2D or 3D bounding box or a geometry.



Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_XMin('BOX3D(1 2 3, 4 5 6)');
st_xmin
-----
1

SELECT ST_XMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmin
-----
1

SELECT ST_XMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmin
-----
-3
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
a BOX3D
SELECT ST_XMin('LINESTRING(1 3, 5 6)');
--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
150406 3)'));
st_xmin
-----
220186.995121892
```

☒☒

`ST_XMax`, `ST_YMax`, `ST_YMin`, `ST_ZMax`, `ST_ZMin`

7.18.11 ST_YMax

`ST_YMax` — Returns the Y maxima of a 2D or 3D bounding box or a geometry.

Synopsis

`float ST_YMax(box3d aGeomorBox2DorBox3D);`

☒☒

Returns the Y maxima of a 2D or 3D bounding box or a geometry.



Note

Although this function is only defined for `box3d`, it also works for `box2d` and geometry values due to automatic casting. However it will not accept a geometry or `box2d` text representation, since those do not auto-cast.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_YMax('BOX3D(1 2 3, 4 5 6)');
st_ymax
-----
5

SELECT ST_YMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_ymax
-----
6

SELECT ST_YMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_ymax
-----
4
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
a BOX3D
SELECT ST_YMax('LINESTRING(1 3, 5 6)');
--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
150406 3)'));
st_ymax
-----
150506.126829327
```


☒☒

`ST_XMin`, `ST_XMax`, `ST_YMin`, `ST_ZMax`, `ST_ZMin`

7.18.12 ST_YMin

`ST_YMin` — Returns the Y minima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_YMin**(box3d aGeomorBox2DorBox3D);

☒☒

Returns the Y minima of a 2D or 3D bounding box or a geometry.



Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_YMin('BOX3D(1 2 3, 4 5 6)');
st_ymin
-----
2

SELECT ST_YMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_ymin
-----
3

SELECT ST_YMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_ymin
-----
2
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
a BOX3D
SELECT ST_YMin('LINESTRING(1 3, 5 6)');
--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
150406 3)'));
st_ymin
-----
150406
```

☒☒

`ST_GeomFromEWKT`, `ST_XMin`, `ST_XMax`, `ST_YMax`, `ST_ZMax`, `ST_ZMin`

7.18.13 ST_ZMax

`ST_ZMax` — Returns the Z maxima of a 2D or 3D bounding box or a geometry.

Synopsis

`float ST_ZMax(box3d aGeomorBox2DorBox3D);`

☒☒

Returns the Z maxima of a 2D or 3D bounding box or a geometry.



Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_ZMax('BOX3D(1 2 3, 4 5 6)');
```

```
st_zmax
```

```
-----
```

```
6
```

```
SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
```

```
st_zmax
```

```
-----
```

```
7
```

```
SELECT ST_ZMax('BOX3D(-3 2 1, 3 4 1)');
```

```
st_zmax
```

```
-----
```

```
1
```

--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D ↩

```
SELECT ST_ZMax('LINESTRING(1 3 4, 5 6 7)');
```

```
--ERROR: BOX3D parser - doesn't start with BOX3D(
```

```
SELECT ST_ZMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'));
```

```
st_zmax
```

```
-----
```

```
3
```

☒☒

`ST_GeomFromEWKT`, `ST_XMin`, `ST_XMax`, `ST_YMax`, `ST_YMin`, `ST_ZMax`

7.18.14 ST_ZMin

`ST_ZMin` — Returns the Z minima of a 2D or 3D bounding box or a geometry.

Synopsis

`float ST_ZMin(box3d aGeomorBox2DorBox3D);`

☒☒

Returns the Z minima of a 2D or 3D bounding box or a geometry.



Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_ZMin('BOX3D(1 2 3, 4 5 6)');
st_zmin
-----
3

SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
st_zmin
-----
4

SELECT ST_ZMin('BOX3D(-3 2 1, 3 4 1) ');
st_zmin
-----
1
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
a BOX3D
SELECT ST_ZMin('LINESTRING(1 3 4, 5 6 7)');
--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
150406 3)')));
st_zmin
-----
1
```

返回

`ST_GeomFromEWKT`, `ST_GeomFromText`, `ST_XMin`, `ST_XMax`, `ST_YMax`, `ST_YMin`, `ST_ZMax`

7.19 线性引用 (Linear Referencing)

7.19.1 ST_LineInterpolatePoint

`ST_LineInterpolatePoint` — Returns a point interpolated along a line at a fractional location.

Synopsis

geometry **ST_LineInterpolatePoint**(geometry a_linestring, float8 a_fraction);
geography **ST_LineInterpolatePoint**(geography a_linestring, float8 a_fraction, boolean use_spheroid = true);

返回

返回一个点，该点位于沿输入线段的指定分数位置。该函数返回的点的类型与输入线段的类型相同。如果输入线段是 2D 的，则返回的点是 2D 的；如果输入线段是 3D 的，则返回的点是 3D 的。如果输入线段是 geography 类型，则返回的点是 geography 类型。该函数的输入参数 `a_fraction` 必须是一个在 0 到 1 之间的浮点值。

该函数与 `ST_LineLocatePoint` 函数一起使用。



Note

This function computes points in 2D and then interpolates values for Z and M, while `ST_3DLineInterpolatePoint` computes points in 3D and only interpolates the M value.



Note

1.1.1 版本中，M 和 Z 值 (如果有) 将被插值。如果输入线段的 `a_fraction` 为 0.0，则返回的点是线段的起始点。

0.8.2 版本中，返回的点是 2D 的。1.1.1 版本中，返回的点是 3D 的。

兼容性: 2.1.0 版本中，该函数名为 `ST_Line_Interpolate_Point`。



This function supports 3d and will not drop the z-index.

图 7.19.1

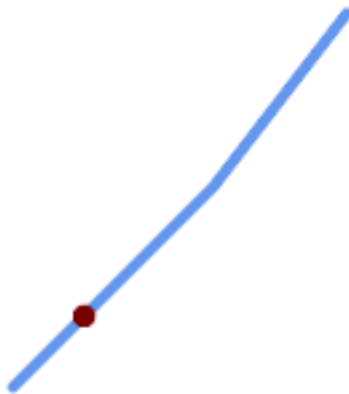


图 7.19.2 20% 的点 (0.20) 沿着一段线

-- The point 20% along a line

```
SELECT ST_AsEWKT( ST_LineInterpolatePoint(
    'LINESTRING(25 50, 100 125, 150 190)',
    0.2 ));
-----
POINT(51.5974135047432 76.5974135047432)
```

The mid-point of a 3D line:

```
SELECT ST_AsEWKT( ST_LineInterpolatePoint('
    LINESTRING(1 2 3, 4 5 6, 6 7 8)',
    0.5 ));
-----
POINT(3.5 4.5 5.5)
```

The closest point on a line to a point:

```
SELECT ST_AsText( ST_LineInterpolatePoint( line.geom,
    ST_LineLocatePoint( line.geom, 'POINT(4 3)')) )
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As geom) AS line;
-----
POINT(3 4)
```

图 7.19.3

[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);

注意

ST_LineInterpolatePoint 返回一个 POINT。如果输入是 LINESTRING，则返回的 POINT 的几何类型是 POINT。如果输入是 GEOMETRY，则返回的 POINT 的几何类型是 GEOMETRY。如果输入是 GEOMETRY，则返回的 POINT 的几何类型是 GEOMETRY。如果输入是 GEOMETRY，则返回的 POINT 的几何类型是 GEOMETRY。



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 版本支持 3D。



This function supports 3d and will not drop the z-index.

注意

Return point 20% along 3D line

```
SELECT ST_AsText(
  ST_3DLineInterpolatePoint('LINESTRING(25 50 70, 100 125 90, 150 190 200)',
    0.20));

st_asetext
-----
POINT Z (59.0675892910822 84.0675892910822 79.0846904776219)
```

注意

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);

注意

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.

图 7.19.3

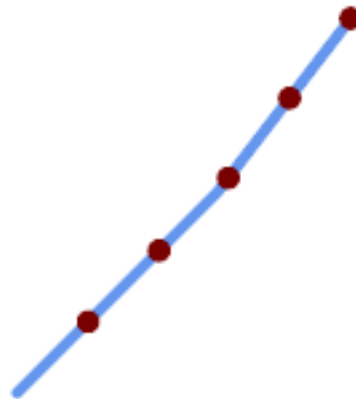


图 7.19.3 *A LineString with points interpolated every 20%*

```
--Return points each 20% along a 2D line
SELECT ST_AsText(ST_LineInterpolatePoints('LINESTRING(25 50, 100 125, 150 190)', 0.20))
-----
MULTIPOINT((51.5974135047432 76.5974135047432),(78.1948270094864 103.194827009486) ↔
, (104.132163186446 130.37181214238),(127.066081593223 160.18590607119),(150 190))
```

图 7.19.4

ST_LineInterpolatePoint, ST_LineLocatePoint

7.19.4 ST_LineLocatePoint

ST_LineLocatePoint — Returns the fractional location of the closest point on a line to a point.

Synopsis

```
float8 ST_LineLocatePoint(geometry a_linestring, geometry a_point);
float8 ST_LineLocatePoint(geography a_linestring, geography a_point, boolean use_spheroid = true);
```

图 7.19.4

图 7.19.4 展示了 ST_LineLocatePoint 函数的使用。该函数返回一个浮点值，表示给定点在直线上的位置。如果点位于直线上，则返回 0 到 1 之间的值；如果点不在直线上，则返回 0 到 1 之间的值，表示最近的点。

图 7.19.4 展示了 ST_LineInterpolatePoint 函数的使用。该函数返回一个点，表示给定点在直线上的位置。如果点位于直线上，则返回该点；如果点不在直线上，则返回最近的点。

图 7.19.4 展示了 ST_LineSubstring 函数的使用。该函数返回一个子字符串，表示给定点在直线上的位置。

1.1.0 版本中，ST_LineLocatePoint 函数返回 0 到 1 之间的值。

图 7.19.4: 2.1.0 版本中，ST_LineLocatePoint 函数返回 0 到 1 之间的值。

例

```
--Rough approximation of finding the street number of a point along the street
--Note the whole foo thing is just to generate dummy data that looks
--like house centroids and street
--We use ST_DWithin to exclude
--houses too far away from the street to be considered on the street
SELECT ST_AsText(house_loc) As as_text_house_loc,
       startstreet_num +
       CAST( (endstreet_num - startstreet_num)
            * ST_LineLocatePoint(street_line, house_loc) As integer) As
       street_num
FROM
  (SELECT ST_GeomFromText('LINESTRING(1 2, 3 4)') As street_line,
        ST_Point(x*1.01,y*1.03) As house_loc, 10 As startstreet_num,
        20 As endstreet_num
  FROM generate_series(1,3) x CROSS JOIN generate_series(2,4) As y)
As foo
WHERE ST_DWithin(street_line, house_loc, 0.2);

as_text_house_loc | street_num
-----+-----
POINT(1.01 2.06) |          10
POINT(2.02 3.09) |          15
POINT(3.03 4.12) |          20

--find closest point on a line to a point or other geometry
SELECT ST_AsText(ST_LineInterpolatePoint(foo.the_line, ST_LineLocatePoint(foo.the_line,
  ST_GeomFromText('POINT(4 3)'))))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As the_line) As foo;
st_astext
-----
POINT(3 4)
```

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);

例

Computes the line which is the section of the input line starting and ending at the given fractional locations. The first argument must be a LINESTRING. The second and third arguments are values in the range [0, 1] representing the start and end locations as fractions of line length. The Z and M values are interpolated for added endpoints if present.

'例' 例 例 例 例 例 例 例 例 **ST_LineInterpolatePoint** 例 例 例 例 例 例.

**Note**

This only works with LINESTRINGs. To use on contiguous MULTILINESTRINGs first join them with **ST_LineMerge**.

**Note**

1.1.1 函数支持 M 和 Z 索引 (3D 和 4D 几何). 1.1.0 版本不支持 M 和 Z 索引.

Enhanced: 3.4.0 - Support for geography was introduced.

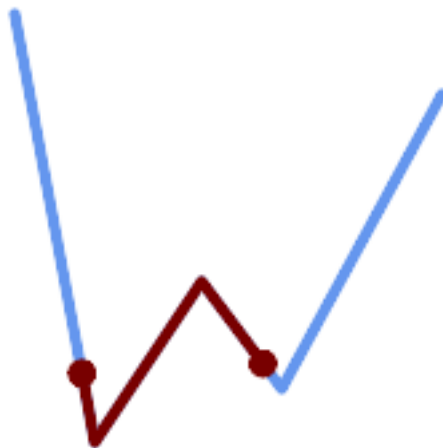
版本: 2.1.0 引入, 在 2.0.x 版本中通过 **ST_Line_Substring** 实现.

1.1.0 版本引入. 1.1.1 版本支持 Z 和 M 索引.



This function supports 3d and will not drop the z-index.

图例



图例 1/3 距离 (0.333, 0.666) 处提取的子字符串

```
SELECT ST_AsText(ST_LineSubstring( 'LINESTRING (20 180, 50 20, 90 80, 120 40, 180 150)',  
0.333, 0.666));
```

```
LINESTRING (45.17311810399485 45.74337011202746, 50 20, 90 80, 112.97593050157862  
49.36542599789519)
```

If start and end locations are the same, the result is a POINT.

```
SELECT ST_AsText(ST_LineSubstring( 'LINESTRING(25 50, 100 125, 150 190)', 0.333, 0.333));
```

```
POINT(69.2846934853974 94.2846934853974)
```

A query to cut a LineString into sections of length 100 or shorter. It uses `generate_series()` with a `CROSS JOIN LATERAL` to produce the equivalent of a FOR loop.

```
WITH data(id, geom) AS (VALUES
    ( 'A', 'LINESTRING( 0 0, 200 0)::geometry ),
    ( 'B', 'LINESTRING( 0 100, 350 100)::geometry ),
    ( 'C', 'LINESTRING( 0 200, 50 200)::geometry )
)
SELECT id, i,
       ST_AsText( ST_LineSubstring( geom, startfrac, LEAST( endfrac, 1 )) ) AS geom
FROM (
    SELECT id, geom, ST_Length(geom) len, 100 sublen FROM data
) AS d
CROSS JOIN LATERAL (
    SELECT i, (sublen * i) / len AS startfrac,
           (sublen * (i+1)) / len AS endfrac
    FROM generate_series(0, floor( len / sublen )::integer ) AS t(i)
    -- skip last i if line length is exact multiple of sublen
    WHERE (sublen * i) / len <
> 1.0
) AS d2;
```

id	i	geom
A	0	LINESTRING(0 0,100 0)
A	1	LINESTRING(100 0,200 0)
B	0	LINESTRING(0 100,100 100)
B	1	LINESTRING(100 100,200 100)
B	2	LINESTRING(200 100,300 100)
B	3	LINESTRING(300 100,350 100)
C	0	LINESTRING(0 200,50 200)

Geography implementation measures along a spheroid, geometry along a line

```
SELECT ST_AsText(ST_LineSubstring( 'LINESTRING(-118.2436 34.0522, -71.0570 42.3611)::' ␣
    geography, 0.333, 0.666),6) AS geog_sub
, ST_AsText(ST_LineSubstring('LINESTRING(-118.2436 34.0522, -71.0570 42.3611)::' ␣
    geometry, 0.333, 0.666),6) AS geom_sub;
-----
geog_sub | LINESTRING(-104.167064 38.854691,-87.674646 41.849854)
geom_sub | LINESTRING(-102.530462 36.819064,-86.817324 39.585927)
```



ST_Length, **ST_LineExtend**, **ST_LineInterpolatePoint**, **ST_LineMerge**

7.19.6 ST_LocateAlong

ST_LocateAlong — Returns the point(s) on a geometry that match a measure value.

Synopsis

geometry **ST_LocateAlong**(geometry geom_with_measure, float8 measure, float8 offset = 0);

函数

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 函数 `ST_Locate_Along_Measure` 返回沿几何体的位置。

更新: 2.0.0 函数 `ST_Locate_Along_Measure` 返回沿几何体的位置。该函数支持 M 坐标。该函数在 2.0.0 版本中引入。



This function supports M coordinates.



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.13

函数

```
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))
```

函数

[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);

说明

Clipping a non-convex POLYGON may produce invalid geometry. The semantic is specified by the *ISO/IEC 13249-3 SQL/MM Spatial* standard.

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 函数 `ST_Locate_Between_Measures` 支持 M 坐标。

2.0.0 函数 `ST_Locate_Along_Measure` 支持 M 坐标。2.0.0 版本中，`ST_Locate_Between_Measures` 支持 M 坐标。

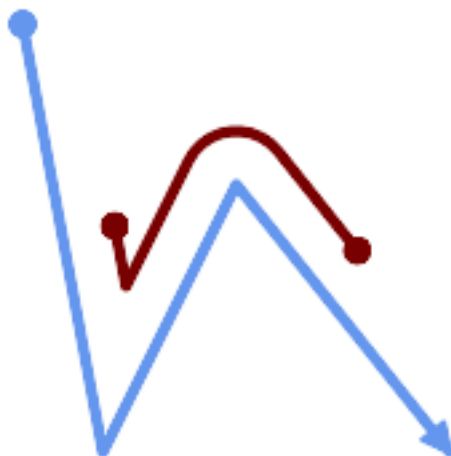
Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE.

✓ This function supports M coordinates.

✓ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

示例

```
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 ↵
82.12248075654186,69.16695286779743 103.05526528559065,82.11145618000168 ↵
128.94427190999915,84.24893681714357 132.32493442618113,87.01636951231555 ↵
135.21267035596549,90.30307285299679 137.49198684843182,93.97759758337769 ↵
139.07172433557758,97.89298381958797 139.8887023914453,101.89263860095893 ↵
139.9102465862721,105.81659870902816 139.13549527600819,109.50792827749828 ↵
137.5954340631298,112.81899532549731 135.351656550512,115.6173761888606 ↵
132.49390095108848,145.31017306064817 95.37790486135405))
```

☒☒

ST_LocateAlong, ST_LocateAlong, ST_LocateBetween

7.19.8 ST_LocateBetweenElevations

ST_LocateBetweenElevations — Returns the portions of a geometry that lie in an elevation (Z) range.

Synopsis

geometry **ST_LocateBetweenElevations**(geometry geom, float8 elevation_start, float8 elevation_end);

☒☒

Returns a geometry (collection) with the portions of a geometry that lie in an elevation (Z) range.

Clipping a non-convex POLYGON may produce invalid geometry.

1.4.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE.

✔ This function supports 3d and will not drop the z-index.

☒☒

```
SELECT ST_AsText(
  ST_LocateBetweenElevations(
    'LINESTRING(1 2 3, 4 5 6)::geometry,
    2, 4 ));
```

st_astext

```
-----
MULTILINESTRING Z ((1 2 3,2 3 4))
```

```
SELECT ST_AsText(
  ST_LocateBetweenElevations(
    'LINESTRING(1 2 6, 4 5 -1, 7 8 9)',
    6, 9)) As ewelev;
```

ewelev

```
-----
GEOMETRYCOLLECTION Z (POINT Z (1 2 6),LINESTRING Z (6.1 7.1 6,7 8 9))
```

函数

[ST_Dump](#), [ST_LocateAlong](#), [ST_LocateBetween](#)

7.19.9 ST_InterpolatePoint

`ST_InterpolatePoint` — 返回线性几何体在给定位置 (M 分量) 的插值。

Synopsis

```
float8 ST_InterpolatePoint(geometry linear_geom_with_measure, geometry point);
```

函数

Returns an interpolated measure value of a linear measured geometry at the location closest to the given point.



Note

Use this function only for linear geometries with an M component

2.0.0 版本引入。



This function supports 3d and will not drop the z-index.

函数

```
SELECT ST_InterpolatePoint('LINESTRING M (0 0 0, 10 0 20)', 'POINT(5 5)');
-----
10
```

函数

[ST_AddMeasure](#), [ST_LocateAlong](#), [ST_LocateBetween](#)

7.19.10 ST_AddMeasure

`ST_AddMeasure` — 在线性几何体上插值。

Synopsis

```
geometry ST_AddMeasure(geometry geom_mline, float8 measure_start, float8 measure_end);
```

注意

此函数在 1.5.0 版本之前不支持 3D 几何。在 1.5.0 版本中，它支持 3D 几何，并且不会丢弃 z 索引。

1.5.0 版本支持 3D 几何。



This function supports 3d and will not drop the z-index.

示例

```
SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0, 2 0, 4 0)'),1,4)) As ewelev;
           ewelev
-----
LINESTRINGM(1 0 1,2 0 2,4 0 4)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;
           ewelev
-----
LINESTRING(1 0 4 10,2 0 4 20,4 0 4 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRINGM(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;
           ewelev
-----
LINESTRINGM(1 0 10,2 0 20,4 0 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('MULTILINESTRINGM((1 0 4, 2 0 4, 4 0 4),(1 0 4, 2 0 4, 4 0 4)'),10,70)) As ←
           ewelev;
           ewelev
-----
MULTILINESTRINGM((1 0 10,2 0 20,4 0 40),(1 0 40,2 0 50,4 0 70))
```

7.20 Trajectory Functions

7.20.1 ST_IsValidTrajectory

ST_IsValidTrajectory — Tests if the geometry is a valid trajectory.

Synopsis

boolean **ST_IsValidTrajectory**(geometry line);

注意

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 版本新增。



This function supports 3d and will not drop the z-index.

例

```
-- A valid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(
  ST_MakePointM(0,0,1),
  ST_MakePointM(0,1,2))
);
t

-- An invalid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0,0,1), ST_MakePointM(0,1,0)));
NOTICE:  Measure of vertex 1 (0) not bigger than measure of vertex 0 (1)
st_isvalidtrajectory
-----
f
```

例

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);

例

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 版本新增。



This function supports 3d and will not drop the z-index.

例


```
-- Return the time in which two objects moving between 10:00 and 11:00
-- are closest to each other and their distance at that point
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)::geometry',
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry',
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) b
), cpa AS (
  SELECT ST_ClosestPointOfApproach(a,b) m FROM inp
), points AS (
  SELECT ST_GeometryN(ST_LocateAlong(a,m),1) pa,
         ST_GeometryN(ST_LocateAlong(b,m),1) pb
  FROM inp, cpa
)
SELECT to_timestamp(m) t,
       ST_3DDistance(pa,pb) distance,
       ST_AsText(pa, 2) AS pa, ST_AsText(pb, 2) AS pb
FROM points, cpa;
```

t	distance	pa
	pb	
2015-05-26 10:45:31.034483-07	1.9652147377620688	POINT ZM (7.59 0 3.79 1432662331.03)
		POINT ZM (9.1 1.24 3.93 1432662331.03)

[ST_IsValidTrajectory](#), [ST_DistanceCPA](#), [ST_LocateAlong](#), [ST_AddMeasure](#)

7.20.3 ST_DistanceCPA

`ST_DistanceCPA` — Returns the distance between the closest point of approach of two trajectories.

Synopsis

`float8 ST_DistanceCPA(geometry track1, geometry track2);`

Returns the distance (in 2D) between two trajectories at their closest point of approach. Inputs must be valid trajectories as checked by [ST_IsValidTrajectory](#). Null is returned if the trajectories do not overlap in their M ranges.

2.2.0

 This function supports 3d and will not drop the z-index.

例

```
-- Return the minimum distance of two objects moving between 10:00 and 11:00
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)::geometry',
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry',
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) b
)
SELECT ST_DistanceCPA(a,b) distance FROM inp;

      distance
-----
1.965214737762069
```

例

[ST_IsValidTrajectory](#), [ST_ClosestPointOfApproach](#), [ST_AddMeasure](#), [ST_DistanceCPA](#)

7.20.4 ST_CPAWithin

ST_CPAWithin — Tests if the closest point of approach of two trajectories is within the specified distance.

Synopsis

boolean **ST_CPAWithin**(geometry track1, geometry track2, float8 dist);

例

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 简体中文。



This function supports 3d and will not drop the z-index.

例

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)::geometry',
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry',
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) b
)
```

```
)
SELECT ST_CPAWithin(a,b,2), ST_DistanceCPA(a,b) distance FROM inp;

 st_cpawithin |      distance
-----+-----
 t            | 1.96521473776207
```



ST_IsValidTrajectory, ST_ClosestPointOfApproach, ST_DistanceCPA, |=|

7.21 Version Functions

7.21.1 PostGIS_Extensions_Upgrade

PostGIS_Extensions_Upgrade — Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.

Synopsis

text **PostGIS_Extensions_Upgrade**(text target_version=null);



Packages and upgrades PostGIS extensions to given or latest version. Only extensions you have installed in the database will be packaged and upgraded if needed. Reports full PostGIS version and build configuration infos after. This is short-hand for doing multiple CREATE EXTENSION .. FROM unpackaged and ALTER EXTENSION .. UPDATE for each PostGIS extension. Currently only tries to upgrade extensions postgis, postgis_raster, postgis_sfcgal, postgis_topology, and postgis_tiger_geocoder. Availability: 2.5.0



Note

Changed: 3.4.0 to add target_version argument.

Changed: 3.3.0 support for upgrades from any PostGIS version. Does not work on all systems.

Changed: 3.0.0 to repackage loose extensions and support postgis_raster.



```
SELECT PostGIS_Extensions_Upgrade();
```

```
NOTICE: Packaging extension postgis
NOTICE: Packaging extension postgis_raster
NOTICE: Packaging extension postgis_sfcgal
NOTICE: Extension postgis_topology is not available or not packagable for some reason
NOTICE: Extension postgis_tiger_geocoder is not available or not packagable for some reason
      reason

      postgis_extensions_upgrade
-----+-----
 Upgrade completed, run SELECT postgis_full_version(); for details
(1 row)
```



Section [3.4](#), [PostGIS_GEOS_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_Wagyu_Version](#), [PostGIS_Version](#)

7.21.2 PostGIS_Full_Version

`PostGIS_Full_Version` — Reports full PostGIS version and build configuration infos.

Synopsis

text `PostGIS_Full_Version()`;



Reports full PostGIS version and build configuration infos. Also informs about synchronization between libraries and scripts suggesting upgrades as needed.

Enhanced: 3.4.0 now includes extra PROJ configurations `NETWORK_ENABLED`, `URL_ENDPOINT` and `DATABASE_PATH` of `proj.db` location



```
SELECT PostGIS_Full_Version();
```

	postgis_full_version

POSTGIS="3.4.0dev 3.3.0rc2-993-g61bdf43a7" [EXTENSION] PGSQL="160" GEOS="3.12.0dev-CAPI ↵	
-1.18.0" SFCGAL="1.3.8" PROJ="7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj. ↵	
org USER_WRITABLE_DIRECTORY=/tmp/proj DATABASE_PATH=/usr/share/proj/proj.db" GDAL="GDAL ↵	
3.2.2, released 2021/03/05" LIBXML="2.9.10" LIBJSON="0.15" LIBPROTOBUF="1.3.3" WAGYU ↵	
="0.5.0 (Internal)" TOPOLOGY RASTER	
(1 row)	



Section [3.4](#), [PostGIS_GEOS_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_Wagyu_Version](#), [PostGIS_Version](#)

7.21.3 PostGIS_GEOS_Version

`PostGIS_GEOS_Version` — Returns the version number of the GEOS library.

Synopsis

text `PostGIS_GEOS_Version()`;



Returns the version number of the GEOS library, or NULL if GEOS support is not enabled.

❏❏

```
SELECT PostGIS_GEOS_Version();
       postgis_geos_version
-----
3.12.0dev-CAPI-1.18.0
(1 row)
```

❏❏

[PostGIS_Full_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_Version](#)

7.21.4 PostGIS_GEOS_Compiled_Version

`PostGIS_GEOS_Compiled_Version` — Returns the version number of the GEOS library against which PostGIS was built.

Synopsis

text **PostGIS_GEOS_Compiled_Version()**;

❏❏

Returns the version number of the GEOS library, or against which PostGIS was built.

Availability: 3.4.0

❏❏

```
SELECT PostGIS_GEOS_Compiled_Version();
       postgis_geos_compiled_version
-----
3.12.0
(1 row)
```

❏❏

[PostGIS_GEOS_Version](#), [PostGIS_Full_Version](#)

7.21.5 PostGIS_Liblwgeom_Version

`PostGIS_Liblwgeom_Version` — Returns the version number of the liblwgeom library. This should match the version of PostGIS.

Synopsis

text **PostGIS_Liblwgeom_Version()**;

函数

Returns the version number of the liblwgeom library/

函数

```
SELECT PostGIS_Liblwgeom_Version();
postgis_liblwgeom_version
-----
3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

函数

[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()`**;

函数

Returns the version number of the LibXML2 library.

1.5 函数参考.

函数

```
SELECT PostGIS_LibXML_Version();
postgis_libxml_version
-----
2.9.10
(1 row)
```

函数

[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()**;

PostGIS_LibJSON_Version()

Returns the version number of the LibJSON-C library.

PostGIS_LibJSON_Version()

```
SELECT PostGIS_LibJSON_Version();
 postgis_libjson_version
-----
0.17
```

PostGIS_LibJSON_Version()

PostGIS_Full_Version, **PostGIS_Lib_Version**, **PostGIS_PROJ_Version**, **PostGIS_GEOS_Version**, **PostGIS_Versions**

7.21.8 PostGIS_Lib_Build_Date

PostGIS_Lib_Build_Date — Returns build date of the PostGIS library.

Synopsis

text **PostGIS_Lib_Build_Date()**;

PostGIS_Lib_Build_Date()

Returns build date of the PostGIS library.

PostGIS_Lib_Build_Date()

```
SELECT PostGIS_Lib_Build_Date();
 postgis_lib_build_date
-----
2023-06-22 03:56:11
(1 row)
```

7.21.9 PostGIS_Lib_Version

PostGIS_Lib_Version — Returns the version number of the PostGIS library.

Synopsis

text **PostGIS_Lib_Version()**;

函数

Returns the version number of the PostGIS library.

函数

```
SELECT PostGIS_Lib_Version();
       postgis_lib_version
-----
       3.4.0dev
(1 row)
```

函数

[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**();

函数

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

函数

```
SELECT PostGIS_PROJ_Version();
       postgis_proj_version
-----
7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org USER_WRITABLE_DIRECTORY=/tmp/ ↵
proj DATABASE_PATH=/usr/share/proj/proj.db
(1 row)
```

函数

[PostGIS_PROJ_Compiled_Version](#), [PostGIS_Full_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_Version](#)

7.21.11 PostGIS_PROJ_Compiled_Version

`PostGIS_PROJ_Compiled_Version` — Returns the version number of the PROJ library against which PostGIS was built.

Synopsis

text **PostGIS_PROJ_Compiled_Version()**;



Returns the version number of the PROJ library, or against which PostGIS was built.

Availability: 3.5.0



```
SELECT PostGIS_PROJ_Compiled_Version();
 postgis_proj_compiled_version
-----
 9.1.1
(1 row)
```



[PostGIS_PROJ_Version](#), [PostGIS_Full_Version](#)

7.21.12 PostGIS_Wagyu_Version

PostGIS_Wagyu_Version — Returns the version number of the internal Wagyu library.

Synopsis

text **PostGIS_Wagyu_Version()**;



Returns the version number of the internal Wagyu library, or NULL if Wagyu support is not enabled.



```
SELECT PostGIS_Wagyu_Version();
 postgis_wagyu_version
-----
 0.5.0 (Internal)
(1 row)
```



[PostGIS_Full_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML2_Version](#), [PostGIS_Version](#)

7.21.13 PostGIS_Scripts_Build_Date

PostGIS_Scripts_Build_Date — Returns build date of the PostGIS scripts.

Synopsis

text **PostGIS_Scripts_Build_Date**();



Returns build date of the PostGIS scripts.

1.0.0RC1 .



```
SELECT PostGIS_Scripts_Build_Date();
       postgis_scripts_build_date
-----
2023-06-22 03:56:11
(1 row)
```



[PostGIS_Full_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_Version](#)

7.21.14 PostGIS_Scripts_Installed

PostGIS_Scripts_Installed — Returns version of the PostGIS scripts installed in this database.

Synopsis

text **PostGIS_Scripts_Installed**();



Returns version of the PostGIS scripts installed in this database.



Note

If the output of this function doesn't match the output of [PostGIS_Scripts_Released](#) you probably missed to properly upgrade an existing database. See the [Upgrading](#) section for more info.

Availability: 0.9.0

❏

```
SELECT PostGIS_Scripts_Installed();
 postgis_scripts_installed
-----
3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

❏

[PostGIS_Full_Version](#), [PostGIS_Scripts_Released](#), [PostGIS_Version](#)

7.21.15 PostGIS_Scripts_Released

`PostGIS_Scripts_Released` — Returns the version number of the `postgis.sql` script released with the installed PostGIS lib.

Synopsis

text **`PostGIS_Scripts_Released()`**;

❏

Returns the version number of the `postgis.sql` script released with the installed PostGIS lib.



Note

Starting with version 1.1.0 this function returns the same value of [PostGIS_Lib_Version](#). Kept for backward compatibility.

Availability: 0.9.0

❏

```
SELECT PostGIS_Scripts_Released();
 postgis_scripts_released
-----
3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

❏

[PostGIS_Full_Version](#), [PostGIS_Scripts_Installed](#), [PostGIS_Lib_Version](#)

7.21.16 PostGIS_Version

`PostGIS_Version` — Returns PostGIS version number and compile-time options.

Synopsis

text **PostGIS_Version()**;

返回

Returns PostGIS version number and compile-time options.

返回

```
SELECT PostGIS_Version();
               postgis_version
-----
3.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
(1 row)
```

返回

PostGIS_Full_Version, **PostGIS_GEOS_Version**, **PostGIS_Lib_Version**, **PostGIS_LibXML_Version**, **PostGIS_PROJ_Version**

7.22 PostGIS GUC(Grand Unified Custom Variable)

7.22.1 postgis.gdal_datapath

postgis.gdal_datapath — GDAL 的 GDAL_DATA 目录的路径。如果未设置，则默认为 GDAL_DATA 目录。

返回

GDAL 的 GDAL_DATA 目录的路径 PostgreSQL GUC 设置。postgis.gdal_datapath 是 GDAL 的 GDAL_DATA 目录的路径。

GDAL 的 GDAL_DATA 目录的路径 (hard-coded) 的路径。GDAL 的 GDAL_DATA 目录的路径。



Note

PostgreSQL 的 postgresql.conf 文件中的 postgis.gdal_datapath 设置。

2.2.0 版本中，GDAL_DATA 目录的路径。



Note

GDAL 的 GDAL_DATA 目录的路径。

❌❌

postgis.gdal_datapath 配置为默认值。

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';
SET postgis.gdal_datapath TO default;
```

配置为 Windows 默认值。

```
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 启用的 GDAL 驱动程序。GDAL 驱动程序 GDAL_SKIP 禁用。

❌❌

PostGIS 启用的 GDAL 驱动程序。GDAL 驱动程序 GDAL_SKIP 禁用。PostgreSQL 配置文件 postgresql.conf 配置。配置为默认值。

PostgreSQL 配置文件 postgresql.conf 配置 POSTGIS_GDAL_ENABLED_DRIVERS 为 (pass) 为 postgis.gdal_enabled_drivers 配置。

配置为默认值。GDAL 驱动程序 GDAL_SKIP 禁用。配置为默认值。配置为默认值。

Note

postgis.gdal_enabled_drivers 配置为默认值。配置为默认值。



- DISABLE_ALL 禁用 GDAL 驱动程序。DISABLE_ALL 禁用, postgis.gdal_enabled_drivers 配置为默认值。
- ENABLE_ALL 启用 GDAL 驱动程序。
- VSICURL 启用 GDAL 驱动程序 /vsicurl/ 配置。

postgis.gdal_enabled_drivers 为 DISABLE_ALL 禁用, DB 配置, ST_FromGDALRaster(), ST_AsGDALRaster(), ST_AsTIFF(), ST_AsJPEG() 和 ST_AsPNG() 配置。



Note

PostGIS 配置, postgis.gdal_enabled_drivers 为 DISABLE_ALL 禁用。

**Note**

GDAL_SKIP 选项 GDAL 的 [Configuration Options](#) 文档。

2.2.0 版本。

要

To set and reset `postgis.gdal_enabled_drivers` for current session

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.gdal_enabled_drivers = default;
```

Set for all new connections to a specific database to specific drivers

```
ALTER DATABASE mygisdb SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

Setting for whole database cluster to enable all drivers. Requires super user access. Also note that database, session, and user settings override this.

```
--writes to postgres.auto.conf
ALTER SYSTEM SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
--Reloads postgres conf
SELECT pg_reload_conf();
```

要

[ST_FromGDALRaster](#), [ST_AsGDALRaster](#), [ST_AsTIFF](#), [ST_AsPNG](#), [ST_AsJPEG](#), [postgis.enable_outdb_raster](#)

7.22.3 postgis.enable_outdb_rasters

`postgis.enable_outdb_rasters` — DB 选项。

要

DB 选项。PostgreSQL 的 `postgresql.conf` 文件。

PostgreSQL 的 `0` 选项 `POSTGIS_ENABLE_OUTDB_RASTERS` (pass) `postgis.enable_outdb_rasters` 选项。

**Note**

`postgis.enable_outdb_rasters` 选项, GUC `postgis.enable_outdb_rasters` 选项。

**Note**

PostGIS 的 `postgis.enable_outdb_rasters` 选项。

2.2.0 版本。

配置参数

`postgis.enable_outdb_rasters` 配置参数。

```
SET postgis.enable_outdb_rasters TO True;
SET postgis.enable_outdb_rasters = default;
SET postgis.enable_outdb_rasters = True;
SET postgis.enable_outdb_rasters = False;
```

Set for all new connections to a specific database

```
ALTER DATABASE gisdb SET postgis.enable_outdb_rasters = true;
```

Setting for whole database cluster. Requires super user access. Also note that database, session, and user settings override this.

```
--writes to postgres.auto.conf
ALTER SYSTEM SET postgis.enable_outdb_rasters = true;
--Reloads postgres conf
SELECT pg_reload_conf();
```

配置参数

`postgis.gdal_enabled_drivers` `postgis.gdal_vsi_options`

7.22.4 postgis.gdal_vsi_options

`postgis.gdal_vsi_options` — DB 配置参数。

配置参数

A string configuration to set options used when working with an out-db raster. **Configuration options** control things like how much space GDAL allocates to local data cache, whether to read overviews, and what access keys to use for remote out-db data sources.

Availability: 3.2.0

配置参数

`postgis.enable_outdb_rasters` 配置参数。

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY= ↵
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyy';
```

Set `postgis.gdal_vsi_options` just for the *current transaction* using the `LOCAL` keyword:

```
SET LOCAL postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxx ↵
AWS_SECRET_ACCESS_KEY=yyyyyyyyyyyyyyyyyyyyyyyyyy';
```

配置参数

`postgis.enable_outdb_rasters` `postgis.gdal_enabled_drivers`

7.22.5 postgis.gdal_cpl_debug

`postgis.gdal_cpl_debug` — A boolean configuration to turn logging of GDAL debug messages on and off.



By default, GDAL logging is printed to stderr, and lower level debug messages are not printed at all. Turning this GUC to true will cause GDAL logging to be sent into the PostgreSQL logging stream, so you can see more or less of it by altering the `client_min_message` PostgreSQL GUC.

Availability: 3.6.0



```
postgis.enable_outdb_rasters postgis.gdal enabled drivers
```

7.23 Troubleshooting Functions

7.23.1 PostGIS_AddBBox

PostGIS_AddBBox — .

Synopsis

```
geometry PostGIS_AddBBox(geometry geomA);
```

[illegible]

Note

1. 本行在 2019 年 12 月 31 日及 2020 年 6 月 30 日，均符合《上市公司治理指引》的要求，不存在违反《上市公司治理指引》的情形。



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 HasBoundingBox

PostGIS HasBBox — `boolean`, `boolean`.

Synopsis

boolean **PostGIS_HasBBox**(geometry geomA);

返回

如果几何体 geomA 具有边界框，则返回 true，否则返回 false。 使用 **PostGIS_AddBBox** 或 **PostGIS_DropBBox** 函数。



This method supports Circular Strings and Curves.

返回

```
SELECT geom
FROM sometable WHERE PostGIS_HasBBox(geom) = false;
```

返回

PostGIS_AddBBox, **PostGIS_DropBBox**

Chapter 8

SFCGAL Functions Reference

SFCGAL 是一个 2D 和 3D 几何处理库，它封装了 CGAL 库的 C++ 接口 (wrapper)。它提供了许多函数，用于处理几何数据，包括点、线、面、体等。

SFCGAL 的官方网站是 <http://www.sfcgal.org>。您可以在该网站上找到 SFCGAL 的文档和下载链接。SFCGAL 依赖于 PostGIS 3.6.1 版本。

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。

 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);

返回

Returns the full version of SFCGAL in use including CGAL and Boost versions

Availability: 3.3.0



This method needs SFCGAL backend.

返回

postgis_sfcgal_version

8.2 SFCGAL Accessors and Setters

8.2.1 CG_ForceLHR

CG_ForceLHR — LHR(Left Hand Reverse; 返回) 返回。

Synopsis

geometry **CG_ForceLHR**(geometry geom);

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).

8.2.2 CG_IsPlanar

CG_IsPlanar — 返回。

Synopsis

boolean **CG_IsPlanar**(geometry geom);

函数

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` — 检查几何体是否是实心体。返回布尔值。

Synopsis

boolean **CG_IsSolid**(geometry geom1);

函数

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，则返回 Polyhedral 表面。否则，返回 NULL。

Synopsis

geometry **CG_MakeSolid**(geometry geom1);

函数

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)。

Synopsis

integer **CG_Orientation**(geometry geom);

返回

返回几何体的朝向。如果几何体是逆时针方向，则返回 -1；如果是顺时针方向，则返回 1。

Availability: 3.5.0



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.

8.2.6 CG_Area

CG_Area — 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))');
      cg_area
-----
      25
(1 row)
```

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 或正浮点数值。

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 或正浮点数值。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 — 3 维几何体的体积。返回 0 或正浮点数值。

Synopsis

```
float CG_Volume(geometry geom1);
```

国国

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)

国国

When closed surfaces are created with WKT, they are treated as areal rather than solid. To make them solid, you need to use **CG_MakeSolid**. Areal geometries have no volume. Here is an example to demonstrate.

```
SELECT CG_Volume(geom) As cube_surface_vol,
       CG_Volume(CG_MakeSolid(geom)) As solid_surface_vol
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
      ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
      ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
      ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
      ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
      ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )':geometry) As f(geom);

cube_surface_vol | solid_surface_vol
-----+-----
0 | 1
```

国国

CG_3DArea, **CG_MakeSolid**, **CG_IsSolid**

8.2.9 ST_ForceLHR

ST_ForceLHR — LHR(Left Hand Reverse; 国国国国) 国国国国国国国国.

Synopsis

```
geometry ST_ForceLHR(geometry geom);
```


¶¶



Warning
`ST_ForceLHR` is deprecated as of 3.5.0. Use `CG_ForceLHR` 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).

8.2.10 ST_IsPlanar

`ST_IsPlanar` — 简体中文.

Synopsis

boolean **ST_IsPlanar**(geometry geom);

¶¶



Warning
`ST_IsPlanar` is deprecated as of 3.5.0. Use `CG_IsPlanar` instead.

2.2.0 简体中文. ¶¶ 2.1.0 简体中文 2.1 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).

8.2.11 ST_IsSolid

`ST_IsSolid` — 简体中文. 简体中文.

Synopsis

boolean **ST_IsSolid**(geometry geom1);



Warning

ST_IsSolid is deprecated as of 3.5.0. Use **CG_IsSolid** instead.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

8.2.12 ST_MakeSolid

ST MakeSolid — 3D model of the part. 3D model of the part. 3D model of the part, 3D model of the part. 3D model of the part. TIN model of the part.

Synopsis

```
geometry ST_MakeSolid(geometry geom1);
```



Warning

Warning
ST_MakeSolid is deprecated as of 3.5.0. Use CG_MakeSolid instead.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

8.2.13 ST Orientation

ST Orientation — ☐☐☐☐☐ (orientation) ☐☐☐☐☐.

Synopsis

```
integer ST_Orientation(geometry geom);
```

函数



Warning

ST_Orientation is deprecated as of 3.5.0. Use **CG_Orientation** instead.

返回几何体的方位角。方位角在 -1 度（北）和 1 度（南）之间。

2.1.0 版本引入。



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.

8.2.14 ST_3DArea

ST_3DArea — 3D 几何体的面积。返回 0 或正数。

Synopsis

```
float ST_3DArea(geometry geom1);
```

函数



Warning

ST_3DArea is deprecated as of 3.5.0. Use **CG_3DArea** instead.

2.1.0 版本引入。



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).

函数

注意: 使用 KWT 函数时，必须使用 SFCGAL 后端。使用 KWT 函数时，必须使用 SFCGAL 后端。

```
SELECT ST_3DArea(geom) As cube_surface_area,
       ST_3DArea(ST_MakeSolid(geom)) As solid_surface_area
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);

cube_surface_area | solid_surface_area
-----+-----
6 | 0
```

☐☐

[ST_Area](#), [ST_MakeSolid](#), [ST_IsSolid](#), [ST_Area](#)

8.2.15 ST_Volume

ST_Volume — 3 维几何体的体积。返回几何体 (多面体) 的体积。

Synopsis

float **ST_Volume**(geometry geom1);

☐☐



Warning

ST_Volume is deprecated as of 3.5.0. Use **CG_Volume** instead.

2.2.0 版本引入。

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 9.1 (same as ST_3DVolume)

☐☐

WKT 多面体, 多面体。使用 [ST_MakeSolid](#) 创建多面体。返回几何体的体积。

```
SELECT ST_Volume(geom) As cube_surface_vol,
       ST_Volume(ST_MakeSolid(geom)) As solid_surface_vol
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);

cube_surface_vol | solid_surface_vol
-----+-----
0 | 1
```

[ST_3DArea](#), [ST_MakeSolid](#), [ST_IsSolid](#)

8.3 SFCGAL Processing and Relationship Functions

8.3.1 CG_Intersection

CG_Intersection — Computes the intersection of two geometries

Synopsis

geometry **CG_Intersection**(geometry geomA , geometry geomB);

Computes the intersection of two geometries.

Performed by the SFCGAL module



Note

NOTE: this function returns a geometry representing the intersection.

Availability: 3.5.0



This method needs SFCGAL backend.

```
SELECT ST_AsText(CG_Intersection('LINESTRING(0 0, 5 5)', 'LINESTRING(5 0, 0 5)'));
      cg_intersection
-----
POINT(2.5 2.5)
(1 row)
```

函数

[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);

函数

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).

函数示例

```
SELECT CG_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
   cg_intersects
   -----
f
(1 row)
SELECT CG_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
   cg_intersects
   -----
t
(1 row)
```

函数

[CG_3DIntersects](#), [ST_3DIntersects](#), [ST_Intersects](#), [ST_Disjoint](#)

8.3.3 CG_3DIntersects

CG_3DIntersects — Tests if two 3D geometries intersect

Synopsis

boolean **CG_3DIntersects**(geometry geomA , geometry geomB);

￼￼

Tests if two 3D geometries intersect. 3D geometries intersect if they have any point in common in the three-dimensional space.

Performed by the SFCGAL module



Note

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Availability: 3.5.0



This method needs SFCGAL backend.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

￼￼￼￼

```
SELECT CG_3DIntersects('POINT(1.2 0.1 0)', 'POLYHEDRALSURFACE(((0 0 0,0.5 0.5 0,1 0 0,1 1 0,0 1 0,0 0 0)),((1 0 0,2 0 0,2 1 0,1 1 0,1 0 0),(1.2 0.2 0,1.2 0.8 0,1.8 0.8 0,1.8 0.2 0,1.2 0.2 0)))');
      cg_3dintersects
-----
t
(1 row)
```

￼￼

CG_Intersects, **ST_3DIntersects**, **ST_Intersects**, **ST_Disjoint**

8.3.4 CG_Difference

CG_Difference — Computes the geometric difference between two geometries

Synopsis

geometry **CG_Difference**(geometry geomA , geometry geomB);

関数

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).

例

```
SELECT ST_AsText(CG_Difference('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'::geometry, 'LINESTRING (↵
(0 0, 2 2)'::geometry));
   cg_difference
-----
POLYGON((0 0,1 0,1 1,0 1,0 0))
(1 row)
```

関数

[ST_3DDifference](#), [ST_Difference](#)

8.3.5 ST_3DDifference

ST_3DDifference — 3D 幾何体間の差分を計算する関数。

Synopsis

geometry **ST_3DDifference**(geometry geom1, geometry geom2);

関数



Warning

[ST_3DDifference](#) is deprecated as of 3.5.0. Use [CG_3DDifference](#) instead.

geom2 返回 geom1 的副本。

2.2.0 版本引入。

- ✔ This method needs SFCGAL backend.
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

8.3.6 CG_3DDifference

CG_3DDifference — 3 维几何体差。

Synopsis

geometry **CG_3DDifference**(geometry geom1, geometry geom2);

返回

geom2 返回 geom1 的副本。

Availability: 3.5.0

- ✔ This method needs SFCGAL backend.
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

返回

PostGIS **ST_AsX3D** 函数返回 3 维几何体的 X3Dom HTML 格式。HTML 格式。

```
SELECT CG_Extrude(ST_Buffer(
  ST_GeomFromText('POINT(100 90)'),
  50, 'quad_segs=1'),0,0,30) AS geom2;
```

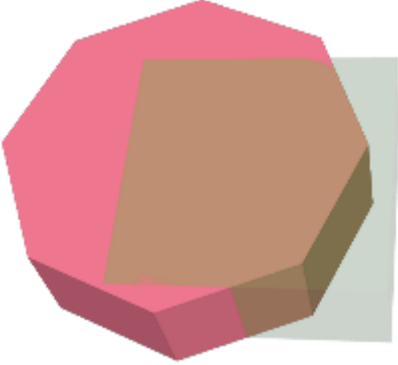
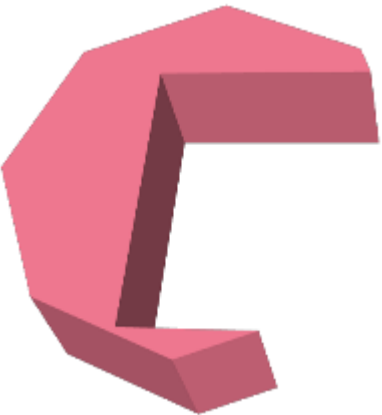


图 3 显示了 *geom2* 的 3D 渲染结果。

```
SELECT CG_3DDifference(geom1,geom2)
  AS geom1,
  CG_Extrude(
    ST_Buffer(ST_GeomFromText('POINT(80 80)'),
    50, 'quad_segs=1'),0,0,30) AS geom2 ) As t;
```



geom2 的 3D 渲染结果。

图 3

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);

图 3

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).

国国国国

```
SELECT CG_Distance('LINESTRING(0.0 0.0,-1.0 -1.0)', 'LINESTRING(3.0 4.0,4.0 5.0)');
      cg_distance
-----
      2.0
(1 row)
```

国国

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);

国国

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).

国国国国

```
SELECT CG_3DDistance('LINESTRING(-1.0 0.0 2.0,1.0 0.0 3.0)', 'TRIANGLE((-4.0 0.0 1.0,4.0 0.0 1.0,0.0 4.0 1.0,-4.0 0.0 1.0))');
      cg_3ddistance
-----
      1
(1 row)
```


CG Distance, ST 3DDistance

8.3.9 ST 3DConvexHull

ST 3DConvexHull — .

Synopsis

```
geometry ST_3DConvexHull(geometry geom1);
```



Warning

Warning
ST_3DConvexHull is deprecated as of 3.5.0. Use CG_3DConvexHull instead.

Availability: 3.3.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

8.3.10 CG 3DConvexHull

CG 3DConvexHull — .

Synopsis

```
geometry CG_3DConvexHull(geometry geom1);
```



Availability: 3.5.0

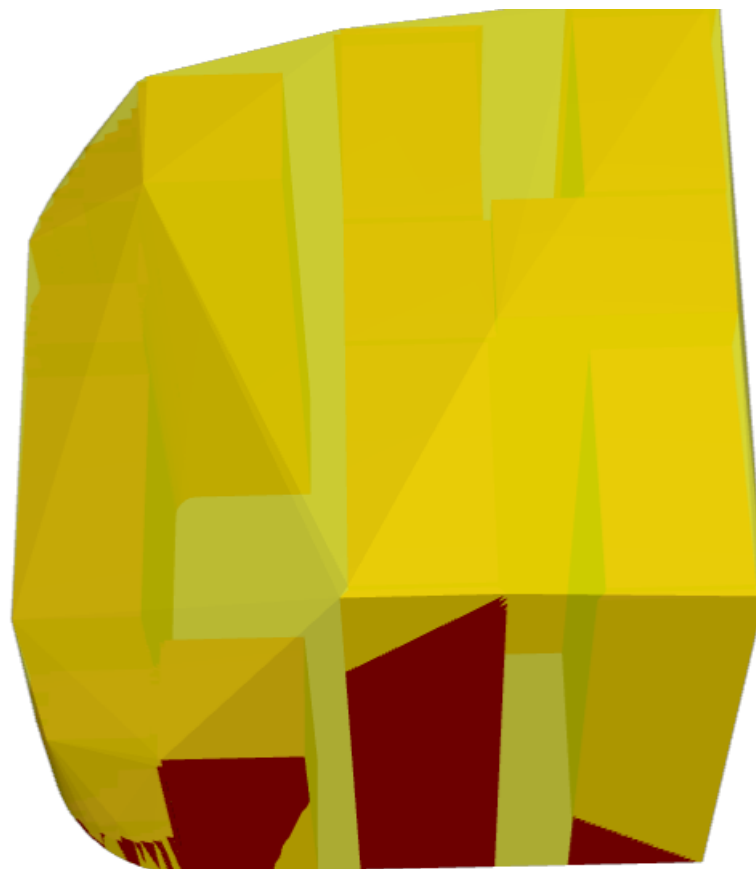
- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

图 8.3.11

```
SELECT ST_AsText(CG_3DConvexHull('LINESTRING Z(0 0 5, 1 5 3, 5 7 6, 9 5 3 , 5 7 5, 6 3 5) ↵
  '::geometry));

POLYHEDRALSURFACE Z (((1 5 3,9 5 3,0 0 5,1 5 3)),((1 5 3,0 0 5,5 7 6,1 5 3)),((5 7 6,5 7 ↵
  5,1 5 3,5 7 6)),((0 0 5,6 3 5,5 7 6,0 0 5)),((6 3 5,9 5 3,5 7 6,6 3 5)),((0 0 5,9 5 3,6 ↵
  3 5,0 0 5)),((9 5 3,5 7 5,5 7 6,9 5 3)),((1 5 3,5 7 5,9 5 3,1 5 3)))

WITH f AS (SELECT i, CG_Extrude(geom, 0,0, i ) AS geom
  FROM ST_Subdivide(ST_Letters('CH'),5) WITH ORDINALITY AS sd(geom,i)
 )
SELECT CG_3DConvexHull(ST_Collect(f.geom) )
FROM f;
```



Original geometry overlaid with 3D convex hull

图 8.3.11

ST_Letters, ST_AsX3D

8.3.11 ST_3DIntersection

ST_3DIntersection — 3 个 3D 几何体的交集。

Synopsis

geometry **ST_3DIntersection**(geometry geom1, geometry geom2);

返回



Warning

ST_3DIntersection is deprecated as of 3.5.0. Use **CG_3DIntersection** instead.

geom1 与 geom2 的 3D 交集。

2.1.0 版本引入。

- ✓ 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 — 3D 交集。

Synopsis

geometry **CG_3DIntersection**(geometry geom1, geometry geom2);

返回

geom1 与 geom2 的 3D 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** 将 3D 几何体转换为 X3Dom HTML 格式。X3Dom HTML 是一种轻量级的 XML 格式，用于在浏览器中显示 3D 模型。

```
SELECT CG_Extrude(ST_Buffer(
  ST_GeomFromText('POINT(100 90)'),
  50, '50',
  CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'),
    50, '50',
    quad_segs=1'),0,0,30) AS geom2;
```

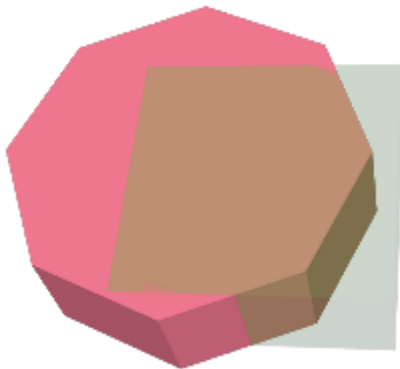
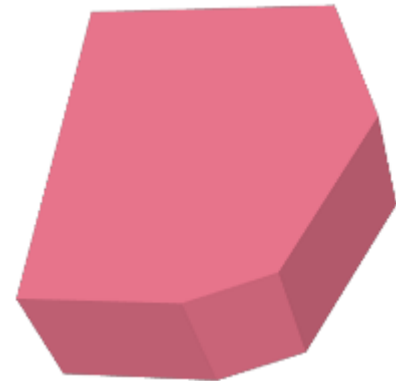


图 3 展示了两个几何体 *geom2* 的交集。

```
SELECT CG_3DIntersection(geom1,geom2)
  50, '50',
  quad_segs=2'),0,0,30) AS geom1,
  quad_segs=2'),0,0,30) AS geom1,
  CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'),
    50, '50',
    quad_segs=1'),0,0,30) AS geom2 ) As t;
```



geom1 与 *geom2* 的交集

3 维几何体的交集

```
SELECT ST_AsText(CG_3DIntersection(linestring, polygon)) As wkt
FROM ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS
linestring
CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

wkt
-----
LINESTRING Z (1 1 8,0.5 0.5 8)
```

图 3 (展示了两个几何体 *geom2* 的交集)

```
SELECT ST_AsText(CG_3DIntersection(
  ST_GeomFromText('POLYHEDRALSURFACE Z( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'),
  'POLYGON Z ((0 0 0, 0 0 0.5, 0 0.5 0.5, 0 0.5 0, 0 0 0))'::geometry))
```

```
TIN Z (((0 0 0,0 0 0.5,0 0.5 0.5,0 0 0)),((0 0.5 0,0 0 0.5,0 0.5 0.5,0 0.5 0)))
```

图 3 展示了两个几何体 *geom2* 的交集 (ST_Dimension 返回 3)

```
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);

国国

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.

国国国国

```
SELECT CG_Union('POINT(.5 0)', 'LINESTRING(-1 0,1 0)');
      cg_union
      -----
LINESTRING(-1 0,0.5 0,1 0)
(1 row)
```


文档

[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);
```

文档



Warning

[ST_3DUnion](#) is deprecated as of 3.5.0. Use [CG_3DUnion](#) instead.

2.2.0 文档.

Availability: 3.3.0 aggregate variant was added

-  This method needs SFCGAL backend.
-  This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
-  This function supports 3d and will not drop the z-index.
-  This function supports Polyhedral surfaces.
-  This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Aggregate variant: returns a geometry that is the 3D union of a rowset of geometries. The ST_3DUnion() function is an “aggregate” function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.

8.3.15 CG_3DUnion

CG_3DUnion — Perform 3D union using postgis_sfcgal.

Synopsis

```
geometry CG_3DUnion(geometry geom1, geometry geom2);
geometry CG_3DUnion(geometry set g1field);
```

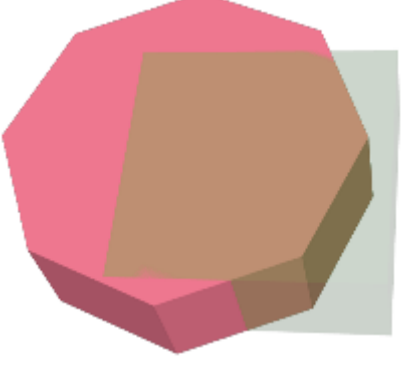
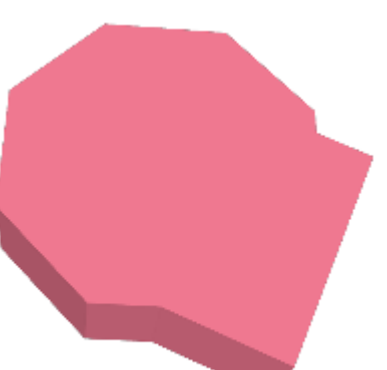

Availability: 3.5.0

- ✔ This method needs SFCGAL backend.
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Aggregate variant: returns a geometry that is the 3D union of a rowset of geometries. The `CG_3DUnion()` function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the `SUM()` and `AVG()` functions do and like most aggregates, it also ignores `NULL` geometries.



PostGIS **ST_AsX3D** 3D **X3Dom** HTML **HTML**.

<pre>SELECT CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 'quad_segs=1'),0,0,30) AS geom2;</pre>  3 3D visualizations of the result of the SQL query. The result is a 3D visualization of the union of the two geometries. The result is a 3D visualization of the union of the two geometries.	<pre>SELECT CG_3DUnion(geom1,geom2)</pre>  geom1 3D visualization of the result of the SQL query. The result is a 3D visualization of the union of the two geometries.

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.

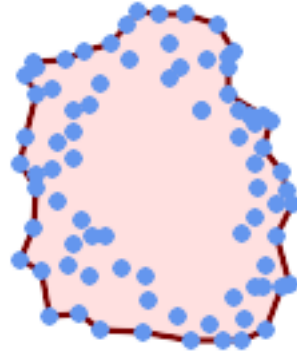
This function effectively computes a concave hull of a geometry in a similar way to **ST_ConcaveHull**, but uses CGAL and a different algorithm.

Availability: 3.5.0 - requires SFCGAL >= 1.4.1.



This method needs SFCGAL backend.

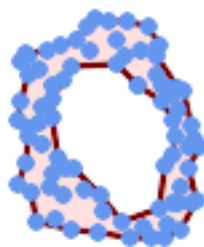
图 10



Alpha-shape of a MultiPoint (same example As [CG_OptimalAlphaShape](#))

```
SELECT ST_AsText(CG_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 ←
61),(32 65),
(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 ←
84),(52 98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 ←
77),(39 88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 ←
86),(60 90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 ←
46),(31 59),(34 86),(45 90),(64 97))'::geometry,80.2));
```

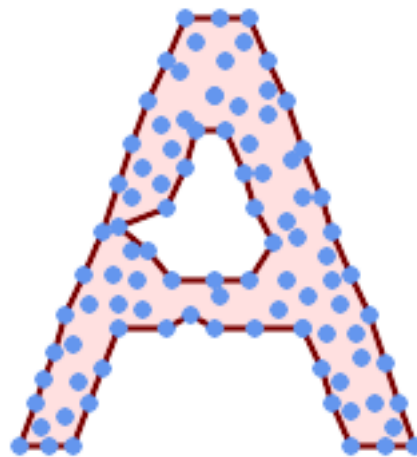
```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,
37 23,30 22,28 33,23 36,26 44,27 54,23 60,24 67,27 77,
24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,
64 97,72 95,76 88,75 84,83 72,85 71,88 58,89 53))
```



Alpha-shape of a MultiPoint, allowing holes (same example as [CG_OptimalAlphaShape](#))

```
SELECT ST_AsText(CG_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70) ←
, (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61) ←
, (32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 84) ←
, (52 98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77) ←
, (39 88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86) ←
, (60 90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 46) ←
, (31 59),(34 86),(45 90),(64 97))'::geometry, 100.1,true))
```

```
POLYGON((89 53,91 50,87 42,90 30,84 19,78 16,73 16,65 16,53 18,43 19,30 22,28 33,23 36,
26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,64 97,72 95,
76 88,75 84,83 72,85 71,88 58,89 53),(36 61,36 68,40 75,43 80,60 81,68 73,77 67,
81 60,82 54,81 47,78 43,76 27,62 22,54 32,44 42,38 46,36 61))
```



Alpha-shape of a MultiPoint, allowing holes (same example as [ST_ConcaveHull](#))

```
SELECT ST_AsText(CG_AlphaShape(
'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 ←
36), (46 20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 ←
100), (63 118), (68 133), (74 149), (81 164), (88 180), (101 180), (112 ←
180), (119 164), (126 149), (132 131), (139 113), (143 100), (150 84), ←
(157 69), (163 51), (168 36), (174 20), (163 20), (150 20), (143 36), ←
(139 49), (132 64), (99 151), (92 138), (88 124), (81 109), (74 93), (70 ←
82), (83 82), (99 82), (112 82), (126 82), (121 96), (114 109), (110 ←
122), (103 138), (99 151), (34 27), (43 31), (48 44), (46 58), (52 73), ←
(63 73), (61 84), (72 71), (90 69), (101 76), (123 71), (141 62), (166 ←
27), (150 33), (159 36), (146 44), (154 53), (152 62), (146 73), (134 ←
76), (143 82), (141 91), (130 98), (126 104), (132 113), (128 127), (117 ←
122), (112 133), (119 144), (108 147), (119 153), (110 171), (103 164), ←
(92 171), (86 160), (88 142), (79 140), (72 124), (83 131), (79 118), ←
(68 113), (63 102), (68 93), (35 45))'::geometry,102.2, true));
```

```
POLYGON((26 20,32 36,35 45,39 55,43 69,50 84,57 100,63 118,68 133,74 149,81 164,88 180,
101 180,112 180,119 164,126 149,132 131,139 113,143 100,150 84,157 69,163 ←
51,168 36,
174 20,163 20,150 20,143 36,139 49,132 64,114 64,99 64,90 69,81 64,63 64,57 ←
49,52 36,46 20,37 20,26 20),
```

```
(74 93,81 109,88 124,92 138,103 138,110 122,114 109,121 96,112 82,99 82,83 82,74 93))
```

国国

[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);

国国

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.

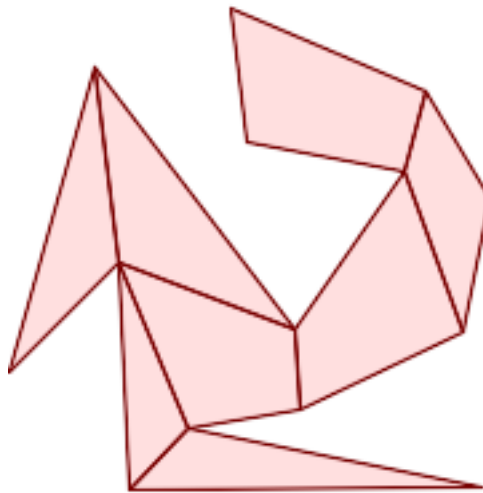
Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.

国国



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)))
```

国国

[CG_YMonotonePartition](#), [CG_GreeneApproxConvexPartition](#), [CG_OptimalConvexPartition](#)

8.3.19 ST_ApproximateMedialAxis

ST_ApproximateMedialAxis — 国国国国国国国国国国国国国国国国.

Synopsis

geometry **ST_ApproximateMedialAxis**(geometry geom);

国国



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 函数参考.

- ✔ 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` — 函数参考.

Synopsis

geometry **CG_ApproximateMedialAxis**(geometry geom);

函数

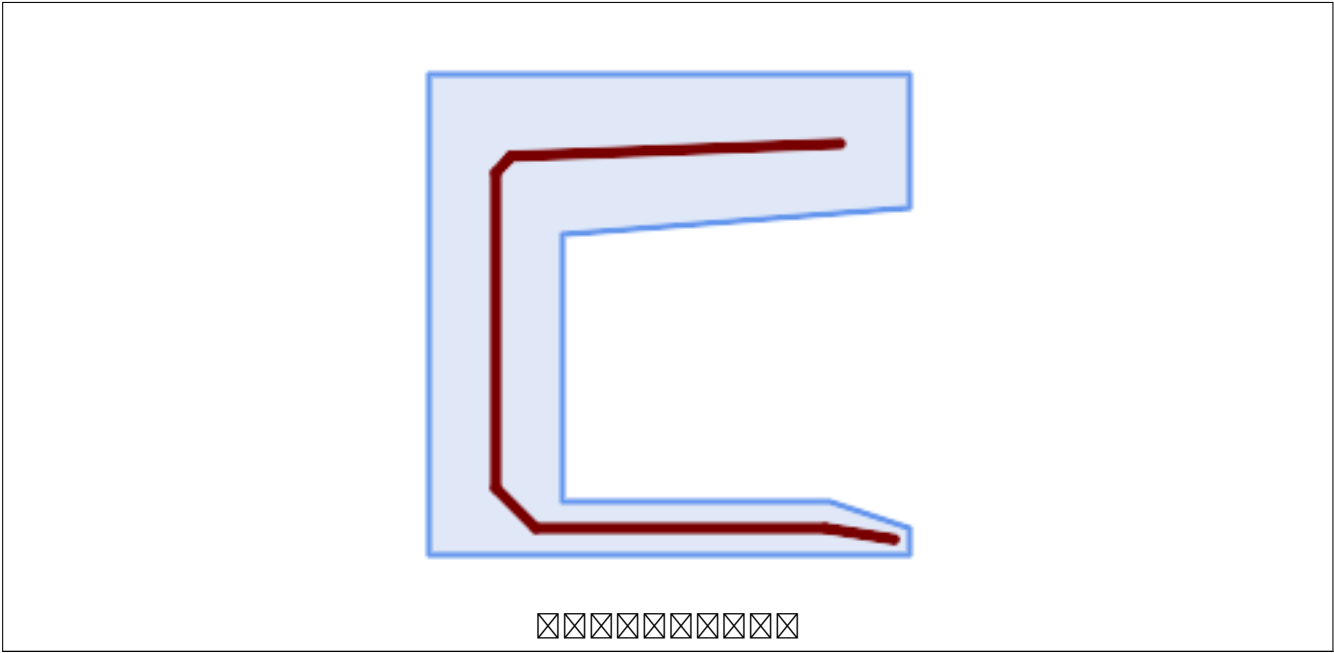
Return an approximate medial axis for the areal input based on its straight skeleton. Uses an SFCGAL specific API when built against a capable version (1.2.0+). Otherwise the function is just a wrapper around `CG_StraightSkeleton` (slower case).

Availability: 3.5.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

函数

```
SELECT CG_ApproximateMedialAxis(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, ↵
190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```

图例

[CG_StraightSkeleton](#), [CG_StraightSkeletonPartition](#)

8.3.21 ST_ConstrainedDelaunayTriangles

ST_ConstrainedDelaunayTriangles — Return a constrained Delaunay triangulation around the given input geometry.

Synopsis

geometry **ST_ConstrainedDelaunayTriangles**(geometry g1);

图例



Warning

[ST_ConstrainedDelaunayTriangles](#) is deprecated as of 3.5.0. Use [CG_ConstrainedDelaunayTriangles](#) instead.

Return a **Constrained Delaunay triangulation** around the vertices of the input geometry. Output is a TIN.



This method needs SFCGAL backend.

2.1.0 简体中文.



This function supports 3d and will not drop the z-index.

8.3.22 CG_ConstrainedDelaunayTriangles

CG_ConstrainedDelaunayTriangles — Return a constrained Delaunay triangulation around the given input geometry.

Synopsis

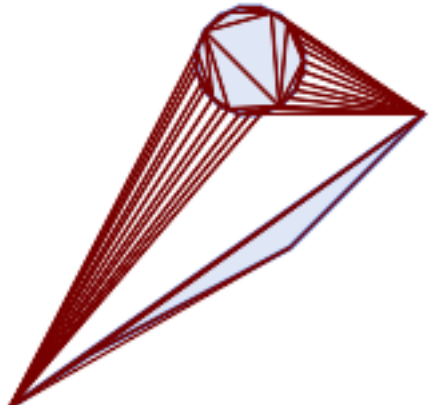
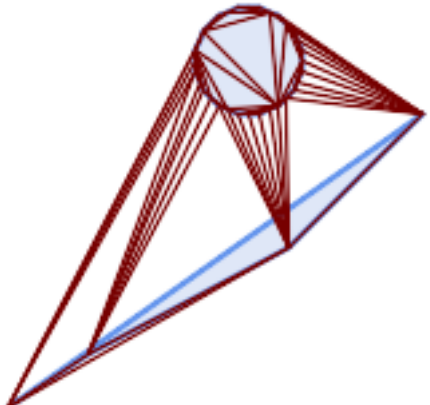
geometry **CG_ConstrainedDelaunayTriangles**(geometry g1);

Return a **Constrained Delaunay triangulation** around the vertices of the input geometry. Output is a TIN.

✔ This method needs SFCGAL backend.

2.1.0

✔ This function supports 3d and will not drop the z-index.

 <i>CG_ConstrainedDelaunayTriangles of 2 polygons</i> <pre>select CG_ConstrainedDelaunayTriangles(ST_Union(POLYGON((175 150, 20 40, 50 60, 125 100, 175 150)), ST_Buffer('POINT(110 170)::geometry', 20)));</pre>	 <i>ST_DelaunayTriangles of 2 polygons. Triangle edges cross polygon boundaries.</i> <pre>select ST_DelaunayTriangles(ST_Union(POLYGON((175 150, 20 40, 50 60, 125 100, 175 150)), ST_Buffer('POINT(110 170)::geometry', 20)));</pre>

函数

[ST_DelaunayTriangles](#), [ST_TriangulatePolygon](#), [CG_Tessellate](#), [ST_ConcaveHull](#), [ST_Dump](#)

8.3.23 ST_Extrude

ST_Extrude — 将几何体沿 z 轴挤出。

Synopsis

geometry **ST_Extrude**(geometry geom, float x, float y, float z);

函数



Warning

ST_Extrude is deprecated as of 3.5.0. Use **CG_Extrude** instead.

2.1.0 函数参考。

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

8.3.24 CG_Extrude

CG_Extrude — 将几何体沿 z 轴挤出。

Synopsis

geometry **CG_Extrude**(geometry geom, float x, float y, float z);

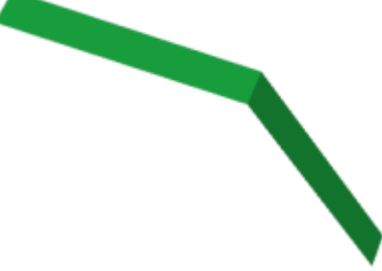
函数

Availability: 3.5.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

图

PostGIS **ST_AsX3D** 返回 3 维几何体的 X3Dom HTML 格式。X3Dom HTML 格式是 HTML 格式。

<pre>SELECT ST_Buffer(ST_GeomFromText('POINT (100 90)'), 50, 'quad_segs=2'),0,0,30);</pre>  图	<pre>CG_Extrude(ST_Buffer(ST_GeomFromText('POINT (100 90)'), 50, 'quad_segs=2'),0,0,30);</pre>  Z 轴 30 个单位高度。 <i>Z(PolyhedralSurfaceZ)</i> 格式。
<pre>SELECT ST_GeomFromText('LINESTRING(50 50, 100 90, 95 150)')</pre>  图	<pre>SELECT CG_Extrude(ST_GeomFromText('LINESTRING(50 50, 100 90, 95 150)')</pre>  Z 轴 30 个单位高度。 <i>Z(PolyhedralSurfaceZ)</i> 格式。

国国

ST_AsX3D, CG_ExtrudeStraightSkeleton

8.3.25 CG_ExtrudeStraightSkeleton

CG_ExtrudeStraightSkeleton — Straight Skeleton Extrusion

Synopsis

geometry **CG_ExtrudeStraightSkeleton**(geometry geom, float roof_height, float body_height = 0);

国国

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.

国国

```
SELECT ST_AsText(CG_ExtrudeStraightSkeleton('POLYGON (( 0 0, 5 0, 5 5, 4 5, 4 4, 0 4, 0 0 ) ←
, (1 1, 1 2, 2 2, 2 1, 1 1))', 3.0, 2.0));
```

```
POLYHEDRALSURFACE Z (((0 0 0,0 4 0,4 4 0,4 5 0,5 5 0,5 0 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 ←
0,1 1 0)),((0 0 0,0 0 2,0 4 2,0 4 0,0 0 0)),((0 4 0,0 4 2,4 4 2,4 4 0,0 4 0)),((4 4 0,4 ←
4 2,4 5 2,4 5 0,4 4 0)),((4 5 0,4 5 2,5 5 2,5 5 0,4 5 0)),((5 5 0,5 5 2,5 0 2,5 0 0,5 5 ←
0)),((5 0 0,5 0 2,0 0 2,0 0 0,5 0 0)),((1 1 0,1 1 2,2 1 2,2 1 0,1 1 0)),((2 1 0,2 1 2,2 ←
2 2,2 2 0,2 1 0)),((2 2 0,2 2 2,1 2 2,1 2 0,2 2 0)),((1 2 0,1 2 2,1 1 2,1 1 0,1 2 0)) ←
,((0.5 2.5 2.5,0 0 2,0.5 0.5 2.5,0.5 2.5 2.5)),((1 3 3,0 4 2,0.5 2.5 2.5,1 3 3)),((0.5 ←
2.5 2.5,0 4 2,0 0 2,0.5 2.5 2.5)),((2.5 0.5 2.5,5 0 2,3.5 1.5 3.5,2.5 0.5 2.5)),((0 0 ←
2.5 0 2,2.5 0.5 2.5,0 0 2)),((0.5 0.5 2.5,0 0 2,2.5 0.5 2.5,0.5 0.5 2.5)),((4.5 3.5 ←
2.5,5 5 2,4.5 4.5 2.5,4.5 3.5 2.5)),((3.5 2.5 3.5,3.5 1.5 3.5,4.5 3.5 2.5,3.5 2.5 3.5)) ←
,((4.5 3.5 2.5,5 0 2,5 5 2,4.5 3.5 2.5)),((3.5 1.5 3.5,5 0 2,4.5 3.5 2.5,3.5 1.5 3.5)) ←
,((5 5 2,4 5 2,4.5 4.5 2.5,5 5 2)),((4.5 4.5 2.5,4 4 2,4.5 3.5 2.5,4.5 4.5 2.5)),((4.5 ←
4.5 2.5,4 5 2,4 4 2,4.5 4.5 2.5)),((3 3 3,0 4 2,1 3 3,3 3 3)),((3.5 2.5 3.5,4.5 3.5 ←
2.5,3 3 3,3.5 2.5 3.5)),((3 3 3,4 4 2,0 4 2,3 3 3)),((4.5 3.5 2.5,4 4 2,3 3 3,4.5 3.5 ←
2.5)),((2 1 2,1 1 2,0.5 0.5 2.5,2 1 2)),((2.5 0.5 2.5,2 1 2,0.5 0.5 2.5,2.5 0.5 2.5)) ←
,((1 1 2,1 2 2,0.5 2.5 2.5,1 1 2)),((0.5 0.5 2.5,1 1 2,0.5 2.5 2.5,0.5 0.5 2.5)),((1 3 ←
3,2 2 2,3 3 3,1 3 3)),((0.5 2.5 2.5,1 2 2,1 3 3,0.5 2.5 2.5)),((1 3 3,1 2 2,2 2 2,1 3 3) ←
),((2 2 2,2 1 2,2.5 0.5 2.5,2 2 2)),((3.5 2.5 3.5,3 3 3,3.5 1.5 3.5,3.5 2.5 3.5)),((3.5 ←
1.5 3.5,2 2 2,2.5 0.5 2.5,3.5 1.5 3.5)),((3 3 3,2 2 2,3.5 1.5 3.5,3 3 3)))
```

国国

[ST_Extrude](#), [CG_ExtrudeStraightSkeleton](#), [CG_StraightSkeleton](#), [CG_StraightSkeletonPartition](#)

8.3.26 CG_GreeneApproxConvexPartition

CG_GreeneApproxConvexPartition — Computes approximal convex partition of the polygon geometry

Synopsis

geometry **CG_GreeneApproxConvexPartition**(geometry geom);

国国

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.

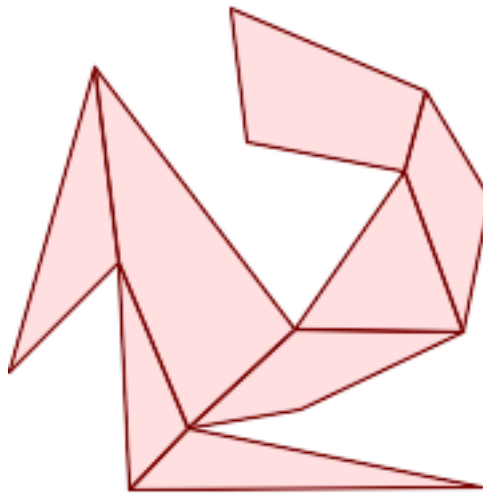
Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.

图 8.3.27



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)))
```

图 8.3.28

[CG_YMonotonePartition](#), [CG_ApproxConvexPartition](#), [CG_OptimalConvexPartition](#)

8.3.27 ST_MinkowskiSum

ST_MinkowskiSum — 计算两个几何体的闵可夫斯基和。

Synopsis

geometry **ST_MinkowskiSum**(geometry geom1, geometry geom2);

图 8.3.29



Warning

ST_MinkowskiSum is deprecated as of 3.5.0. Use [CG_MinkowskiSum](#) instead.

返回两个几何体的闵可夫斯基和。如果输入是点，则返回连接两个点的线段。

该函数使用 A 和 B 的闵可夫斯基和来计算 A 和 B 的闵可夫斯基和。该函数使用运动规划 (motion planning) 和 CAD(computer-aided design) 来计算闵可夫斯基和。该函数使用 [Wikipedia Minkowski addition](#) 来计算闵可夫斯基和。

该函数返回两个几何体的闵可夫斯基和 (点、线、面) 的闵可夫斯基和。如果输入是点，则返回连接两个点的线段。如果输入是线，则返回两个线段的闵可夫斯基和。如果输入是面，则返回两个面的闵可夫斯基和。

该函数使用 [CGAL 2D Minkowskisum](#) 来计算闵可夫斯基和。

2.1.0 版本引入。



This method needs SFCGAL backend.

8.3.28 CG_MinkowskiSum

CG_MinkowskiSum — 计算两个几何体的闵可夫斯基和。

Synopsis

```
geometry CG_MinkowskiSum(geometry geom1, geometry geom2);
```

返回

返回两个几何体的闵可夫斯基和。如果输入是点，则返回连接两个点的线段。

该函数使用 A 和 B 的闵可夫斯基和来计算 A 和 B 的闵可夫斯基和。该函数使用运动规划 (motion planning) 和 CAD(computer-aided design) 来计算闵可夫斯基和。该函数使用 [Wikipedia Minkowski addition](#) 来计算闵可夫斯基和。

该函数返回两个几何体的闵可夫斯基和 (点、线、面) 的闵可夫斯基和。如果输入是点，则返回连接两个点的线段。如果输入是线，则返回两个线段的闵可夫斯基和。如果输入是面，则返回两个面的闵可夫斯基和。

该函数使用 [CGAL 2D Minkowskisum](#) 来计算闵可夫斯基和。

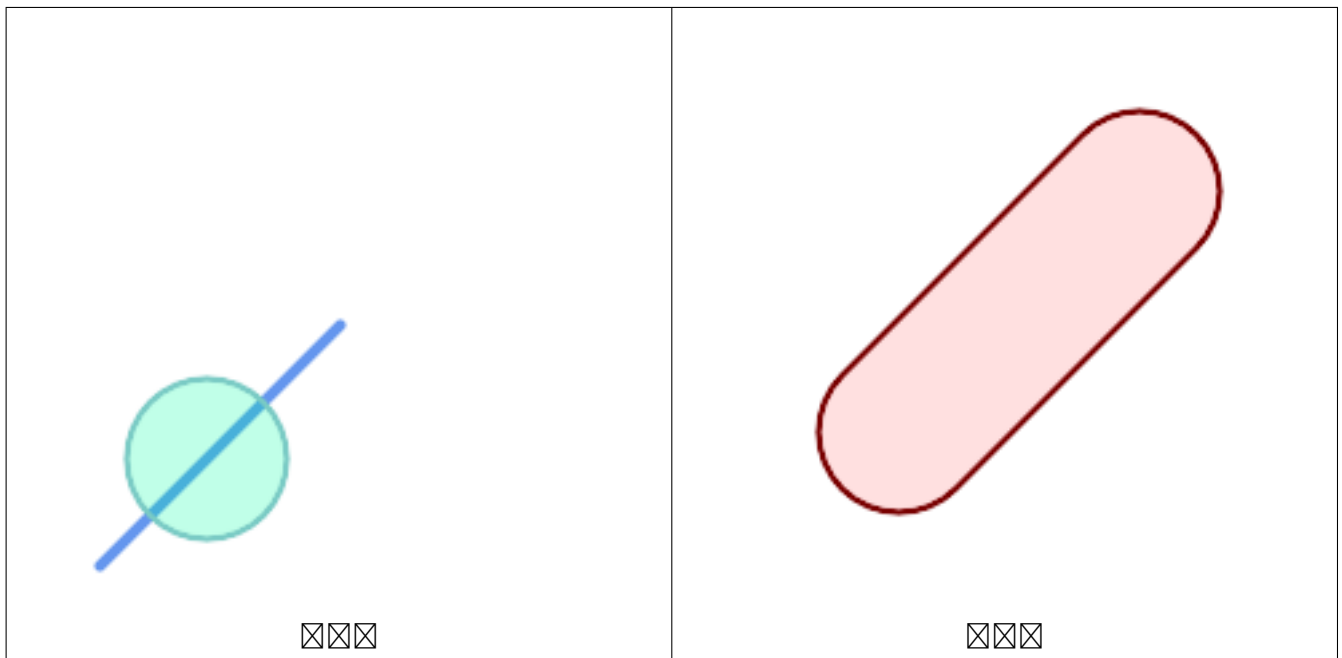
Availability: 3.5.0



This method needs SFCGAL backend.

返回

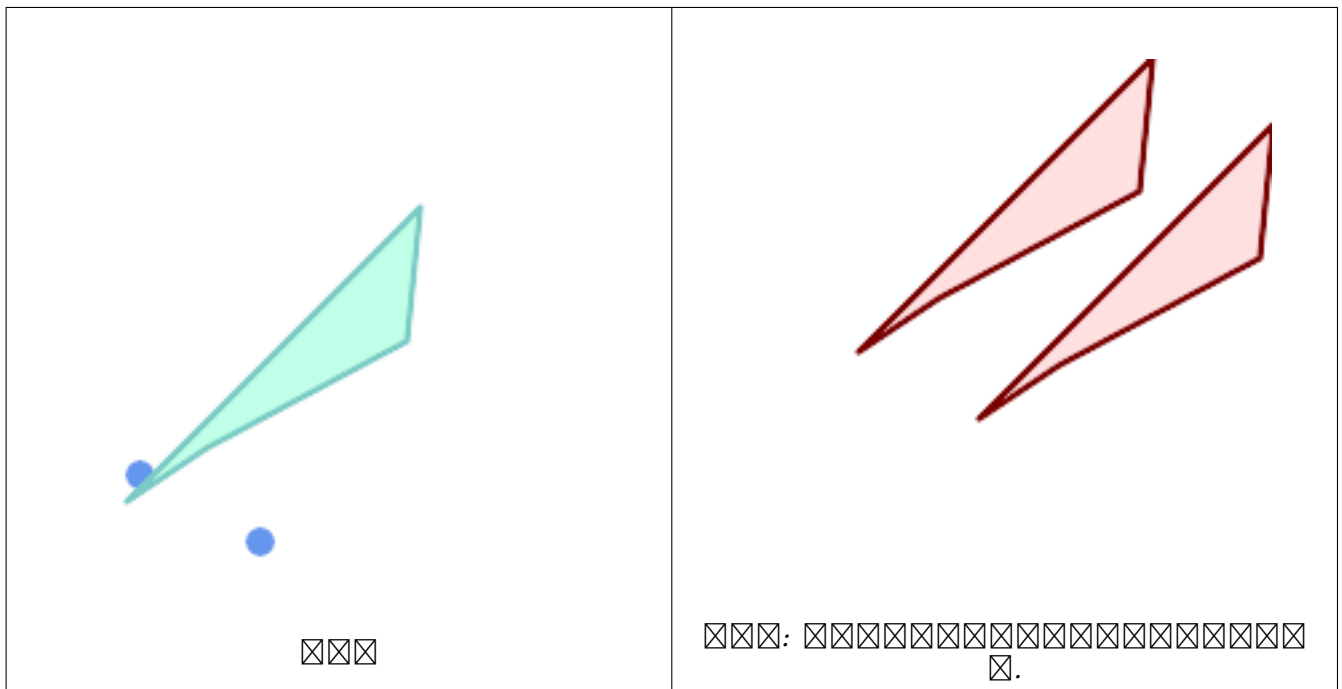
返回两个几何体的闵可夫斯基和。如果输入是点，则返回连接两个点的线段。



```
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 ↵
  30.5764415879031,60.0000000000000 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)))
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXX



```
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 70 115)),
  ((120 65,150 85,225 125,275 175,120 65))
)
```

8.3.29 ST_OptimalAlphaShape

ST_OptimalAlphaShape — Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.

Synopsis

geometry **ST_OptimalAlphaShape**(geometry geom, boolean allow_holes = false, integer nb_components = 1);

Figure 8.3.29



Warning

ST_OptimalAlphaShape is deprecated as of 3.5.0. Use **CG_OptimalAlphaShape** instead.

Computes the "optimal" alpha-shape of the points in a geometry. The alpha-shape is computed using a value of α chosen so that:

1. the number of polygon elements is equal to or smaller than `nb_components` (which defaults to 1)
2. all input points are contained in the shape

The result will not contain holes unless the optional `allow_holes` argument is specified as true.

Availability: 3.3.0 - requires SFCGAL $\geq$ 1.4.1.



This method needs SFCGAL backend.

8.3.30 CG_OptimalAlphaShape

`CG_OptimalAlphaShape` — Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.

Synopsis

geometry **CG_OptimalAlphaShape**(geometry geom, boolean allow_holes = false, integer nb_components = 1);



Computes the "optimal" alpha-shape of the points in a geometry. The alpha-shape is computed using a value of α chosen so that:

1. the number of polygon elements is equal to or smaller than `nb_components` (which defaults to 1)
2. all input points are contained in the shape

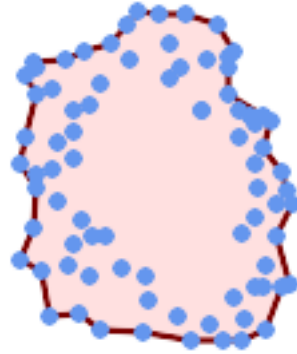
The result will not contain holes unless the optional `allow_holes` argument is specified as true.

Availability: 3.5.0 - requires SFCGAL $\geq$ 1.4.1.



This method needs SFCGAL backend.

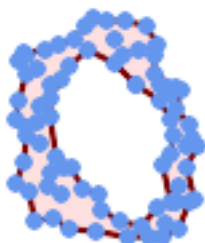
图 10



Optimal alpha-shape of a MultiPoint (same example as [CG_AlphaShape](#))

```
SELECT ST_AsText(CG_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ←
    ,(81 70),
    (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ←
    ,(36 61),(32 65),
    (81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
    (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29) ←
    ,(27 84),(52 98),(72 95),(85 71),
    (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97) ←
    ,(27 77),(39 88),(60 81),
    (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64) ←
    ,(69 86),(60 90),(50 86),(43 80),(36 73),
    (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16) ←
    ,(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry));

POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ←
    33,23 36,
    26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,64 ←
    97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53))
```



Optimal alpha-shape of a MultiPoint, allowing holes (same example as [CG_AlphaShape](#))

```
SELECT ST_AsText(CG_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ↵
, (81 70),(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ↵
, (36 61),(32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50), ↵
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 84) ↵
, (52 98),(72 95),(85 71), ↵
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77) ↵
, (39 88),(60 81), ↵
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86) ↵
, (60 90),(50 86),(43 80),(36 73), ↵
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 46) ↵
, (31 59),(34 86),(45 90),(64 97))'::geometry, allow_holes => true));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ↵
33,23 36,26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 ↵
97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53),(36 61,36 68,40 75,43 ↵
80,50 86,60 81,68 73,77 67,81 60,82 54,81 47,78 43,81 29,76 27,70 20,62 22,55 26,54 ↵
32,48 34,44 42,38 46,36 61))
```



[ST_ConcaveHull](#), [CG_AlphaShape](#)

8.3.31 CG_OptimalConvexPartition

CG_OptimalConvexPartition — Computes an optimal convex partition of the polygon geometry

Synopsis

geometry **CG_OptimalConvexPartition**(geometry geom);



Computes an optimal convex partition of the polygon geometry.

**Note**

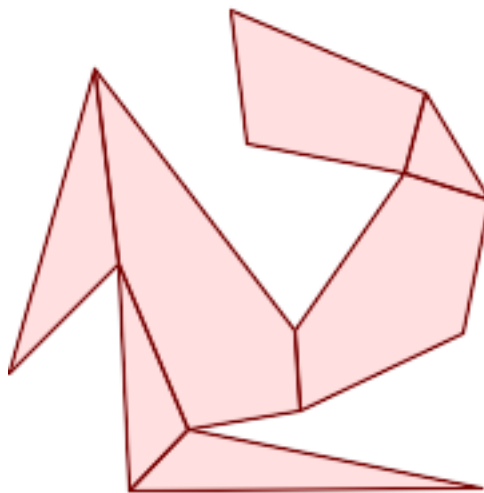
A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. `CG_OptimalConvexPartition` produces a partition that is optimal in the number of pieces.

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.



Optimal Convex Partition (same example As [CG_YMonotonePartition](#), [CG_ApproxConvexPartition](#) and [CG_GreeneApproxConvexPartition](#))

```
SELECT ST_AsText(CG_OptimalConvexPartition('POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150))'::geometry));
```

```
GEOMETRYCOLLECTION(POLYGON((156 150,83 181,89 131,148 120,156 150)),POLYGON((32 159,0 45,41 86,32 159)),POLYGON((45 1,177 2,67 24,45 1)),POLYGON((41 86,45 1,67 24,41 86)),POLYGON((107 61,32 159,41 86,67 24,109 31,107 61)),POLYGON((148 120,107 61,109 31,170 60,180 110,148 120)),POLYGON((156 150,148 120,180 110,156 150)))
```



[CG_YMonotonePartition](#), [CG_ApproxConvexPartition](#), [CG_GreeneApproxConvexPartition](#)

8.3.32 CG_StraightSkeleton

`CG_StraightSkeleton` — 国国国国国国国国国国 (straight skeleton) 国国国国国国.

Synopsis

geometry **CG_StraightSkeleton**(geometry geom, boolean use_distance_as_m = false);

国国

Availability: 3.5.0

Requires SFCGAL >= 1.3.8 for option use_distance_as_m

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

国国

```
SELECT CG_StraightSkeleton(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, 190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```

```
SELECT ST_AsText(CG_StraightSkeleton('POLYGON((0 0,1 0,1 1,0 1,0 0))', true);
```

```
MULTILINESTRING M ((0 0 0,0.5 0.5 0.5),(1 0 0,0.5 0.5 0.5),(1 1 0,0.5 0.5 0.5),(0 1 0,0.5 0.5 0.5));
```

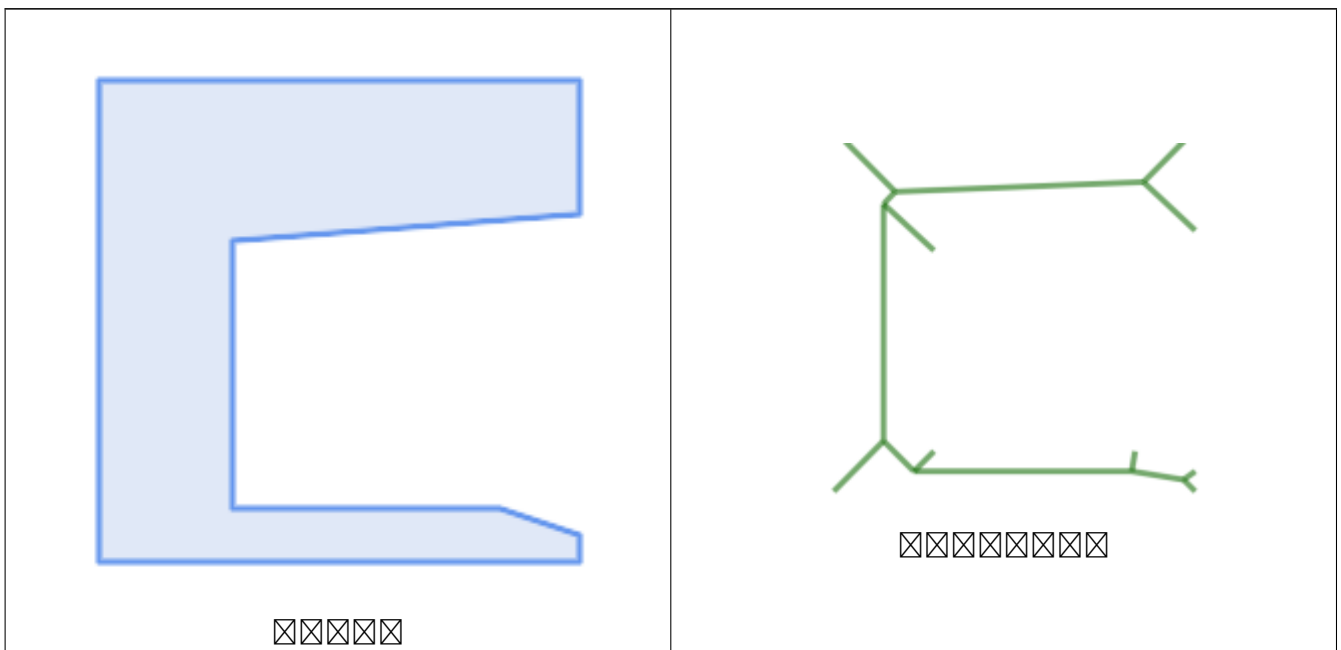
Note that valid inputs with rings that touch at a single point will raise an error.

```
SELECT CG_StraightSkeleton(
'POLYGON((0 0, 3 0, 3 3, 0 3, 0 0), (0 0, 1 2, 2 1, 0 0))');
```

NOTICE: During straight_skeleton(A) :

```
NOTICE: with A: POLYGON((0/1 0/1,3/1 0/1,3/1 3/1,0/1 3/1,0/1 0/1),(0/1 0/1,1/1 2/1,2/1 1/1,0/1 0/1))
```

ERROR: straight skeleton of Polygon with point touching rings is not implemented.



函数

[CG_StraightSkeletonPartition](#), [CG_ExtrudeStraightSkeleton](#)

8.3.33 ST_StraightSkeleton

ST_StraightSkeleton — 计算 (straight skeleton) 骨架。

Synopsis

geometry **ST_StraightSkeleton**(geometry geom);

函数



Warning

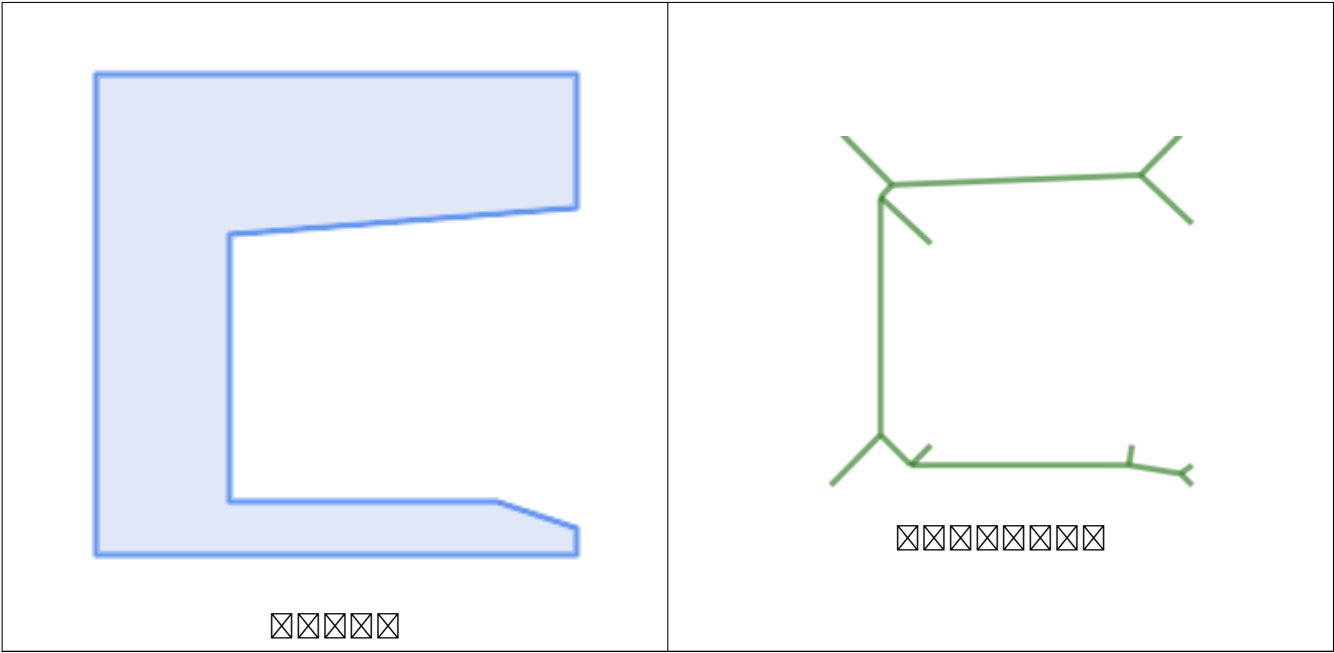
[ST_StraightSkeleton](#) is deprecated as of 3.5.0. Use [CG_StraightSkeleton](#) instead.

2.1.0 版本引入。

- ✓ This method needs SFCGAL backend.
- ✓ This function supports 3d and will not drop the z-index.
- ✓ This function supports Polyhedral surfaces.
- ✓ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

函数

```
SELECT ST_StraightSkeleton(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, 190
20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```

图例

`CG_ExtrudeStraightSkeleton`

8.3.34 ST_Tessellate

ST_Tessellate — 将多边形分解为三角形 (tessellation) 并返回三角形集合 (TIN) 或三角形集合 (TIN) 的集合。

Synopsis

geometry **ST_Tessellate**(geometry geom);

图例



Warning
`ST_Tessellate` is deprecated as of 3.5.0. Use `CG_Tessellate` instead.

[图例] 将多边形分解为三角形 (tessellation) 并返回三角形集合 (TIN) 或三角形集合 (TIN) 的集合。



Note
`ST_TriangulatePolygon` does similar to this function except that it returns a geometry collection of polygons instead of a TIN and also only works with 2D geometries.

2.1.0 版本引入。

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

8.3.35 CG_Tessellate

`CG_Tessellate` — 对几何体进行三角化 (tessellation) 并返回 TIN 格式的结果。

Synopsis

geometry **CG_Tessellate**(geometry geom);

返回

[geometry] 对输入几何体进行三角化并返回 TIN 格式的结果。



Note

ST_TriangulatePolygon 与这个函数类似，但它返回一个几何体集合，而不是 TIN，并且只适用于 2D 几何体。

Availability: 3.5.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

返回

```
SELECT ST_GeomFromText('POLYHEDRALSURFACE
  Z( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
      ((0 0
0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1
0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
      ((0 1
0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1
```



四面体

```
SELECT CG_Tessellate(ST_GeomFromText('
POLYHEDRALSURFACE Z( ((0 0 0, 0 0 1, 0 1 1, 0 1
      ((0 0 0,
0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1
      ((1 1 0,
1 1 1, 1 0 1, 1 0 0, 1 1 0)),
      ((0 1 0,
0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1
```

ST_AsText 结果:

```
TIN Z( ((0 0 0,0 0 1,0 1 1,0 0 0)),((0 1
1 0 0,0 1 0,0 1 1,0 1 0)),
      ((0 0 0,0 1 0,1 1
0,0 0 0)),
      ((1 0 0,0 0 0,1 1
0,1 0 1)),((0 1 1,0 1 0,1 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)))
```



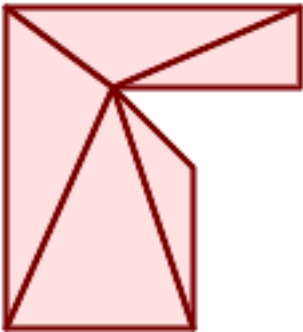
四面体 (带颜色)

```
SELECT 'POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160, 120 160, 120 190, 10 190 ))' AS geometry;
```



多边形

```
SELECT
    CG_Tessellate('
POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160,
;
ST_AsText 多边形:
'::geometry;
    TIN((80 130,50 160,80 70,80 130)),((50 160,10 190,10 70,50 160)),
        ((80 70,50 160,10 70,80 70)),((120 160,120 190,50 160,120 160)),
        ((120 190,10 190,50 160,120 190)))
```



三角化多边形

多边形

[CG_ConstrainedDelaunayTriangles](#), [ST_DelaunayTriangles](#), [ST_TriangulatePolygon](#)

8.3.36 CG_Triangulate

CG_Triangulate — Triangulates a polygonal geometry

Synopsis

geometry **CG_Triangulate**(geometry geom);

多边形

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.

文档

```
SELECT CG_Triangulate('POLYGON((0.0 0.0,1.0 0.0,1.0 1.0,0.0 1.0,0.0 0.0),(0.2 0.2,0.2 0.8,0.8 0.8,0.8 0.2,0.2 0.2))');
      cg_triangulate
-----
      TIN(((0.8 0.2,0.2 0.2,1 0,0.8 0.2)),((0.2 0.2,0 0,1 0,0.2 0.2)),((1 1,0.8 0.8,0.8 0.2,1 1)),((0 1,0 0,0.2 0.2,0 1)),((0 1,0.2 0.8,1 1,0 1)),((0 1,0.2 0.2,0.2 0.8,0 1)),((0.2 0.8,0.8 0.8,1 1,0.2 0.8)),((0.2 0.8,0.2 0.2,0.8 0.8)),((1 1,0.8 0.2,1 0,1 1)),((0.8 0.8,0.2 0.8,0.8 0.2,0.8 0.8)))
(1 row)
```

文档

[CG_ConstrainedDelaunayTriangles](#), [ST_DelaunayTriangles](#), [ST_TriangulatePolygon](#)

8.3.37 CG_Visibility

CG_Visibility — Compute a visibility polygon from a point or a segment in a polygon geometry

Synopsis

```
geometry CG_Visibility(geometry polygon, geometry point);
geometry CG_Visibility(geometry polygon, geometry pointA, geometry pointB);
```

文档

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.

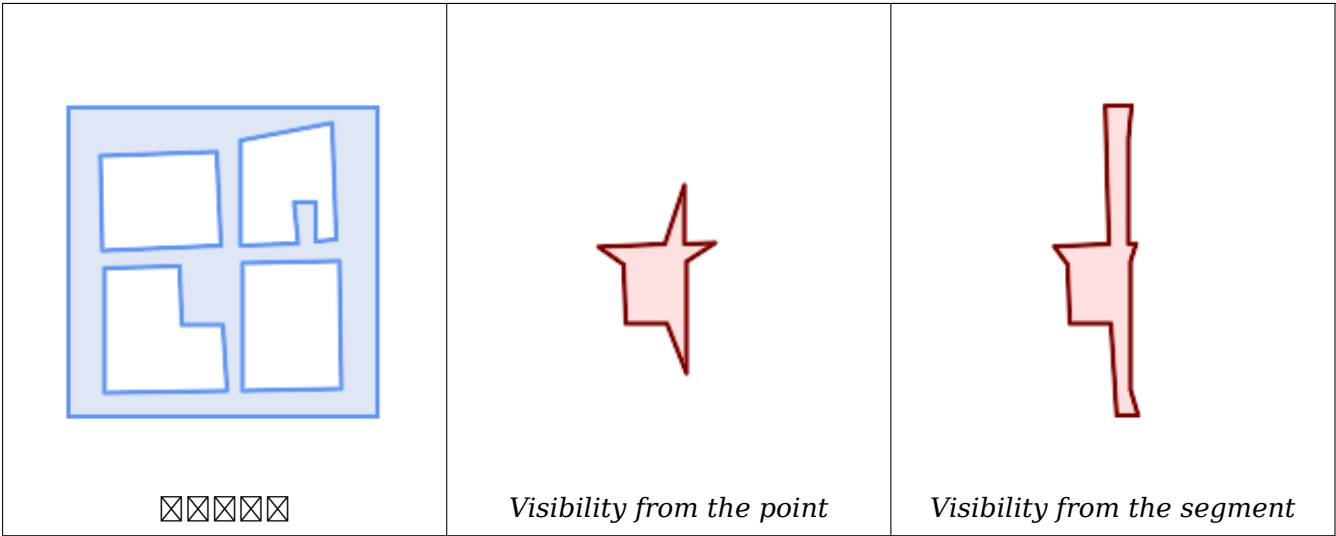


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒

```
SELECT CG_Visibility('POLYGON((23.5 23.5,23.5 173.5,173.5 173.5,173.5 23.5,23.5 23.5),(108 98,108 36,156 37,155 99,108 98),(107 157.5,107 106.5,135 107.5,133 127.5,143.5 127.5,143.5 108.5,153.5 109.5,151.5 166,107 157.5),(41 95.5,41 35,100.5 36,98.5 68,78.5 68,77.5 96.5,41 95.5),(39 150,40 104,97.5 106.5,95.5 152,39 150))'::geometry, 'POINT(91 87)'::geometry);

SELECT CG_Visibility('POLYGON((23.5 23.5,23.5 173.5,173.5 173.5,173.5 23.5,23.5 23.5),(108 98,108 36,156 37,155 99,108 98),(107 157.5,107 106.5,135 107.5,133 127.5,143.5 127.5,143.5 108.5,153.5 109.5,151.5 166,107 157.5),(41 95.5,41 35,100.5 36,98.5 68,78.5 68,77.5 96.5,41 95.5),(39 150,40 104,97.5 106.5,95.5 152,39 150))'::geometry, 'POINT(78.5 68)'::geometry, 'POINT(98.5 68)'::geometry);
```



8.3.38 CG_YMonotonePartition

CG_YMonotonePartition — Computes y-monotone partition of the polygon geometry

Synopsis

```
geometry CG_YMonotonePartition(geometry geom);
```

☒☒

Computes y-monotone partition of the polygon geometry.

Note

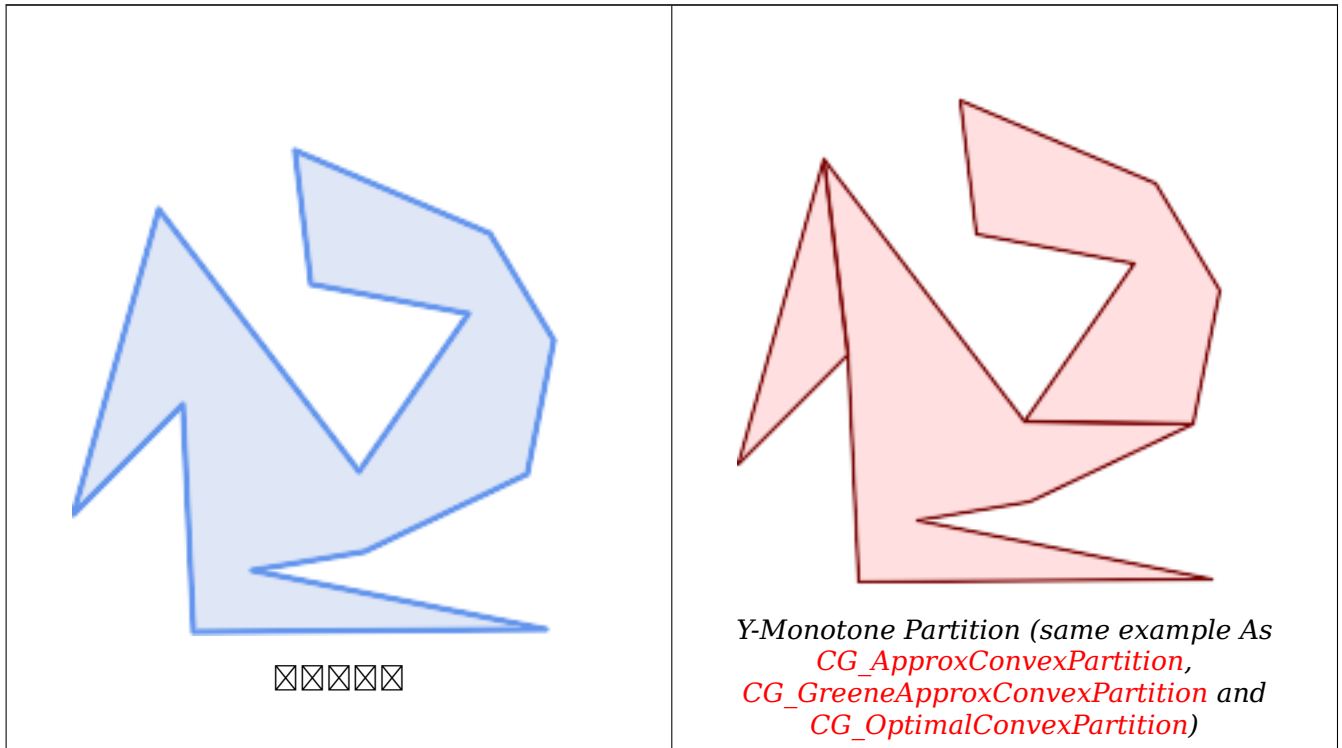
A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. A y-monotone polygon is a polygon whose vertices $v_1, \dots, v_n$ can be divided into two chains $v_1, \dots, v_k$ and $v_k, \dots, v_n, v_1$, such that any horizontal line intersects either chain at most once. This algorithm does not guarantee a bound on the number of polygons produced with respect to the optimal number.

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.



```
SELECT ST_AsText(CG_YMonotonePartition('POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150))'::geometry));
```

```
GEOMETRYCOLLECTION(POLYGON((32 159,0 45,41 86,32 159)),POLYGON((107 61,32 159,41 86,45 1,177 2,67 24,109 31,170 60,107 61)),POLYGON((156 150,83 181,89 131,148 120,107 61,170 60,180 110,156 150)))
```



CG_ApproxConvexPartition, *CG_GreeneApproxConvexPartition*, *CG_OptimalConvexPartition*

8.3.39 CG_StraightSkeletonPartition

`CG_StraightSkeletonPartition` — Computes the straight skeleton partition of a polygon.

Synopsis

geometry **CG_StraightSkeletonPartition**(geometry geom, boolean auto_orientation);

国国

Computes the straight skeleton partition of the input polygon geometry *geom*. The straight skeleton is a partitioning of the polygon into faces formed by tracing the collapse of its edges. If *auto_orientation* is set to true, the function will automatically adjust the orientation of the input polygon to ensure correct results.

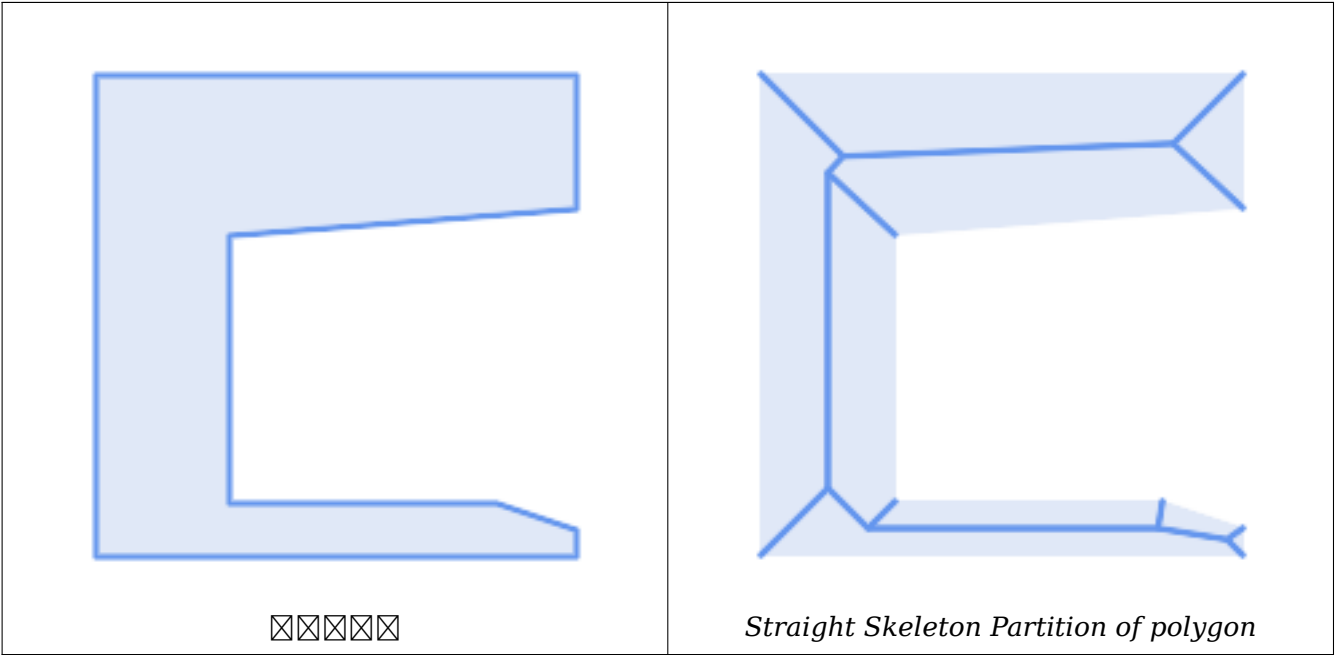
Availability: 3.6.0 - requires SFCGAL >= 2.0.0.

✔ This method needs SFCGAL backend.

国国

```
SELECT ST_AsText(CG_StraightSkeletonPartition('POLYGON((0 0, 4 0, 2 2, 0 0))', true));
-- Result: MULTIPOLYGON(((0 0,2 0.83,2 2)),((4 0,2 0.83,0 0)),((2 2,2 0.83,4 0)))
```

```
SELECT CG_StraightSkeletonPartition(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, 190 20
, 160 30, 60 30, 60 130, 190 140, 190 190 ))')
, true );
```



国国

CG_StraightSkeleton, ST_Polygonize

8.3.40 CG_3DBuffer

CG_3DBuffer — Computes a 3D buffer around a geometry.

Synopsis

geometry **CG_3DBuffer**(geometry geom, float8 radius, integer segments, integer buffer_type);



Generates a 3D buffer around the input geometry *geom* with a specified *radius*. The buffer is constructed in 3D space, creating a volumetric representation of the geometry's surroundings. The *segments* parameter defines the number of segments used to approximate the curved sections of the buffer, with a minimum value of 4 segments required. The *buffer_type* specifies the type of buffer to create: 0: Rounded buffer (default) 1: Flat buffer 2: Square buffer

Input geometry must be a Point or LineString.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



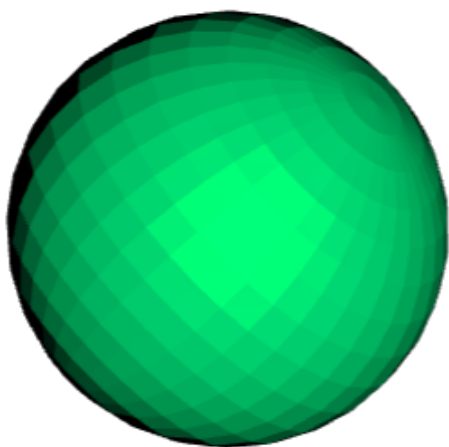
This method needs SFCGAL backend.



```
SELECT ST_AsText(CG_3DBuffer('POINT(0 0 0)', 1, 8, 0));
-- Result: POLYHEDRALSURFACE Z (((0 0 1, 0.5 -0.5 0.71, 0 -0.71 0.71, 0 0 1)), ... )
```

The following images were rendered pasting the output of the ST_AsX3D query into [X3D Viewer](#).

```
SELECT string_agg('<Shape
>' || ST_AsX3D(cgbuffer3d_output) || '<Appearance>
      <Material diffuseColor="0 0.8 0.2" specularColor="0 1 0"/>
      </Appearance>
    </Shape>
>', '');
```



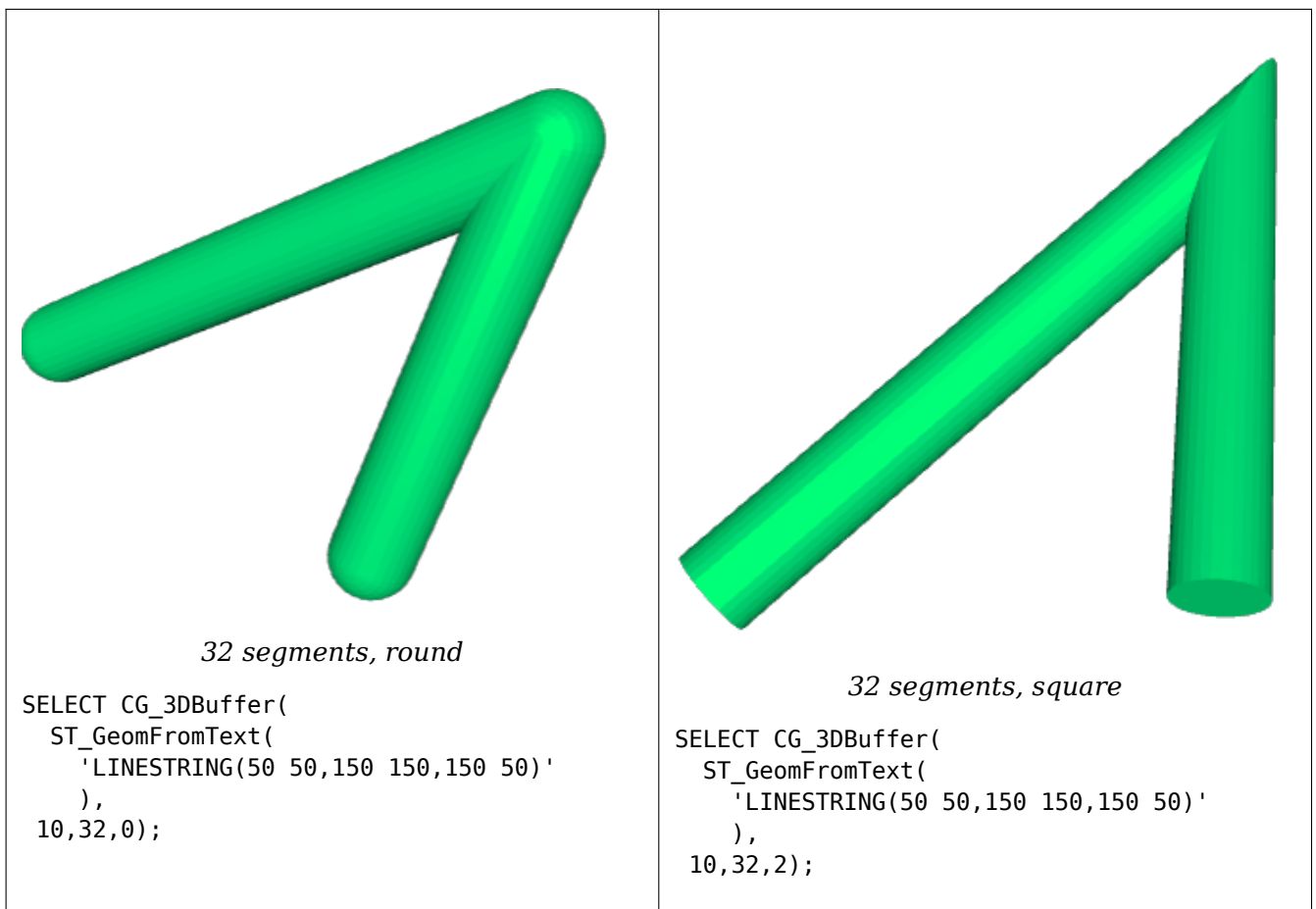
segments=32 (rounded buffer)

```
SELECT CG_3DBuffer(ST_GeomFromText('POINT ↵
(100 90)'), 50,32,0);
```



5 segments rounded

```
SELECT CG_3DBuffer(
  ST_GeomFromText('POINT(100 90)'),
  50,5,0);
```



￼￼

[ST_Buffer](#), [ST_3DConvexHull](#), [ST_AsX3D](#)

8.3.41 CG_Rotate

CG_Rotate — Rotates a geometry by a given angle around the origin (0,0).

Synopsis

geometry **CG_Rotate**(geometry geom, float8 angle);

￼￼

Rotates the input geometry *geom* by *angle* radians around the origin point (0,0). The rotation is performed in 2D space; Z coordinates are not modified. Positive angles rotate the geometry counter-clockwise.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.

❏

```
SELECT ST_AsText(CG_Rotate('LINESTRING(1 0, 0 1)', pi()/2));  
-- Result: LINESTRING(0 1, -1 0)
```

❏

[CG_2DRotate](#), [ST_Rotate](#)

8.3.42 CG_2DRotate

CG_2DRotate — Rotates a geometry by a given angle around a specified point in 2D.

Synopsis

geometry **CG_2DRotate**(geometry geom, float8 angle, float8 cx, float8 cy);

❏

Rotates the input geometry *geom* by *angle* radians around the point (*cx*, *cy*). The rotation is performed in 2D space; Z coordinates are dropped. Positive angles rotate the geometry counter-clockwise.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.

❏

```
SELECT ST_AsText(CG_2DRotate('POINT(1 0)', pi()/2, 1, 1));  
-- Result: POINT(2 1)
```

❏

[CG_Rotate](#), [CG_3DRotate](#)

8.3.43 CG_3DRotate

CG_3DRotate — Rotates a geometry in 3D space around an axis vector.

Synopsis

geometry **CG_3DRotate**(geometry geom, float8 angle, float8 ax, float8 ay, float8 az);

☒☒

Rotates the input geometry *geom* by *angle* radians around an axis defined by the vector (*ax*, *ay*, *az*) passing through the origin (0,0,0).

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.

☒☒

```
SELECT ST_AsText(CG_3DRotate('POINT(1 0 0)', pi()/2, 0, 0, 1));
-- Result: POINT(0 1 0)
```

☒☒

CG_RotateX, **CG_RotateY**, **CG_RotateZ**

8.3.44 CG_RotateX

CG_RotateX — Rotates a geometry around the X-axis by a given angle.

Synopsis

geometry **CG_RotateX**(geometry geom, float8 angle);

☒☒

Rotates the input geometry *geom* by *angle* radians around the X-axis.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.

☒☒

```
SELECT ST_AsText(CG_RotateX('POINT(0 1 0)', pi()/2));
-- Result: POINT(0 0 1)
```

☒☒

CG_RotateY, **CG_RotateZ**, **CG_3DRotate**

8.3.45 CG_RotateY

CG_RotateY — Rotates a geometry around the Y-axis by a given angle.

Synopsis

geometry **CG_RotateY**(geometry geom, float8 angle);

⊠

Rotates the input geometry *geom* by *angle* radians around the Y-axis passing.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.

⊠

```
SELECT ST_AsText(CG_RotateY('POINT(1 0 0)', pi()/2));
-- Result: POINT(0 0 -1)
```

⊠

[CG_RotateX](#), [CG_RotateZ](#), [CG_3DRotate](#)

8.3.46 CG_RotateZ

CG_RotateZ — Rotates a geometry around the Z-axis by a given angle.

Synopsis

geometry **CG_RotateZ**(geometry geom, float8 angle);

⊠

Rotates the input geometry *geom* by *angle* radians around the Z-axis.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.

⊠

```
SELECT ST_AsText(CG_RotateZ('POINT(1 0 0)', pi()/2));
-- Result: POINT(0 1 0)
```

国国

[CG_RotateX](#), [CG_RotateY](#), [CG_3DRotate](#)

8.3.47 CG_Scale

CG_Scale — Scales a geometry uniformly in all dimensions by a given factor.

Synopsis

geometry **CG_Scale**(geometry geom, float8 factor);

国国

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.

国国

```
SELECT ST_AsText(CG_Scale('LINESTRING(1 1, 2 2)', 2));  
-- Result: LINESTRING(2 2, 4 4)
```

国国

[CG_3DScale](#), [CG_3DScaleAroundCenter](#), [ST_Scale](#)

8.3.48 CG_3DScale

CG_3DScale — Scales a geometry by separate factors along X, Y, and Z axes.

Synopsis

geometry **CG_3DScale**(geometry geom, float8 factorX, float8 factorY, float8 factorZ);

国国

Scales the input geometry *geom* by different factors along the X, Y, and Z axes. The scaling is performed relative to the origin point (0,0,0).

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.

❏

```
SELECT ST_AsText(CG_3DScale('POINT(1 1 1)', 2, 3, 4));
-- Result: POINT(2 3 4)
```

❏

[CG_Scale](#), [CG_3DScaleAroundCenter](#)

8.3.49 CG_3DScaleAroundCenter

CG_3DScaleAroundCenter — Scales a geometry in 3D space around a specified center point.

Synopsis

geometry **CG_3DScaleAroundCenter**(geometry geom, float8 factorX, float8 factorY, float8 factorZ, float8 centerX, float8 centerY, float8 centerZ);

❏

Scales the input geometry *geom* by different factors along the X, Y, and Z axes, relative to a specified center point (*centerX*, *centerY*, *centerZ*).

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.

❏

```
SELECT ST_AsText(CG_3DScaleAroundCenter('POINT(2 2 2)', 0.5, 0.5, 0.5, 1, 1, 1));
-- Result: POINT(1.5 1.5 1.5)
```

❏

[CG_Scale](#), [CG_3DScale](#)

8.3.50 CG_Translate

CG_Translate — Translates (moves) a geometry by given offsets in 2D space.

Synopsis

geometry **CG_Translate**(geometry geom, float8 deltaX, float8 deltaY);

☒☒

Translates the input geometry *geom* by adding *deltaX* to the X coordinates and *deltaY* to the Y coordinates. Z coordinates are dropped.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.

☒☒

```
SELECT ST_AsText(CG_Translate('LINESTRING(1 1, 2 2)', 1, -1));
-- Result: LINESTRING(2 0, 3 1)
```

☒☒

CG_3DTranslate, **ST_Translate**

8.3.51 CG_3DTranslate

CG_3DTranslate — Translates (moves) a geometry by given offsets in 3D space.

Synopsis

geometry **CG_3DTranslate**(geometry geom, float8 deltaX, float8 deltaY, float8 deltaZ);

☒☒

Translates the input geometry *geom* by adding *deltaX* to the X coordinates, *deltaY* to the Y coordinates, and *deltaZ* to the Z coordinates.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.

☒☒

```
SELECT ST_AsText(CG_3DTranslate('POINT(1 1 1)', 1, -1, 2));
-- Result: POINT(2 0 3)
```

☒☒

CG_Translate, **ST_Translate**

8.3.52 CG_Simplify

CG_Simplify — Reduces the complexity of a geometry while preserving essential features and Z/M values.

Synopsis

geometry **CG_Simplify**(geometry geom, double precision threshold, boolean preserveTopology = false);

⚠

Simplifies a geometry using SFCGAL's simplification algorithm, which reduces the number of points or vertices while preserving the essential features of the geometry. This function preserves Z and M values during simplification.

The algorithm is based on constrained triangulation and uses the [CGAL Polyline Simplification 2](#) library with additional handling to preserve Z and M coordinates. When topology is preserved and geometries intersect, Z and M values are interpolated at intersection points.

This function works well with 3D terrain-like geometries (2.5D) but is not designed for vertical surfaces like walls.

Availability: 3.6.0 - requires SFCGAL >= 2.1.0



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.



This function supports M coordinates.

Parameters

geom Input geometry

threshold Maximum distance threshold (in geometry unit) for simplification. The higher this value, the more simplified the resulting geometry will be.

preserveTopology If set to true, the function ensures that the topology of the geometry is preserved. When geometries intersect in this mode, Z and M values at intersection points are interpolated. The default value is false.

Return Value

Returns a simplified geometry with preserved Z and M values.

⚠

```
-- Simplify a polygon with a threshold of 0.5
SELECT ST_AsText(CG_Simplify(ST_GeomFromText('POLYGON((0 0, 0 1, 0.1 1, 0.2 1, 0.3 1, 0.4 1, 0.5 1, 1 1, 1 0, 0 0))'), 0.5));

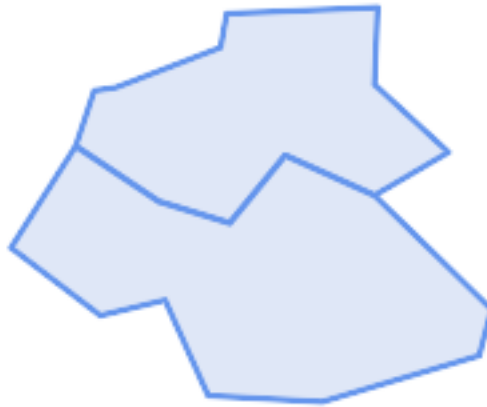
-- Simplify a 3D terrain geometry while preserving topology and Z values
SELECT ST_AsText(CG_Simplify(ST_GeomFromText('LINESTRING Z(0 0 0, 0 1 1, 0.1 1 1, 0.2 1 1, 0.3 1 1, 1 1 2)'), 0.2, true));

-- Simplify a geometry with both Z and M values
SELECT ST_AsText(CG_Simplify(ST_GeomFromText('LINESTRING ZM(0 0 0 1, 0 1 1 2, 0.1 1 1 3, 0.2 1 1 4, 0.3 1 1 5, 1 1 2 6)'), 0.2));
```

```
-- Simplify two geometry together preserving Z and M values, without topology
SELECT ST_AsText(CG_Simplify('GEOMETRYCOLLECTION ZM(LINESTRING ZM(-1 -1 3 4, 0 0 10 100, 1
1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150), POLYGON ZM((0 0 10 100, 1
1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150, 0 0 10 100)))', 2, false));

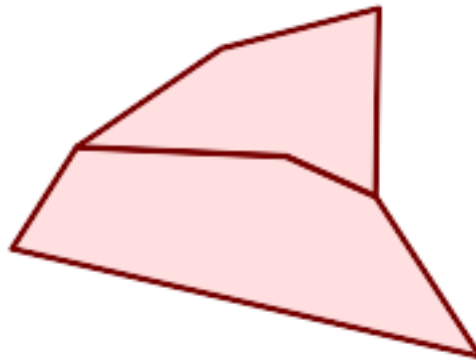
-- Simplify two geometry together preserving Z and M values, with topology
SELECT ST_AsText(CG_Simplify('GEOMETRYCOLLECTION ZM(LINESTRING ZM(-1 -1 3 4, 0 0 10 100, 1
1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150), POLYGON ZM((0 0 10 100, 1
1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150, 0 0 10 100)))', 2, true));

WITH depts_pds as (SELECT ST_GeomFromText('GEOMETRYCOLLECTION(
POLYGON((88.46 158.85,90.77 171.54,147.31 173.85,146.15 145,173.85 119.62,146.15
103.46,112.69 118.46,91.92 93.08,65.38 101.15,34.23 121.92,41.15 142.69,49.23
143.85,88.46 158.85)),
POLYGON((112.69 118.46,146.15 103.46,190 60.77,185.38 43.46,126.54 26.15,83.85 28.46,67.69
64.23,43.46 58.46,10 83.85,34.23 121.92,65.38 101.15,91.92 93.08,112.69 118.46)))
') as geom)
SELECT geom FROM depts_pds;
```



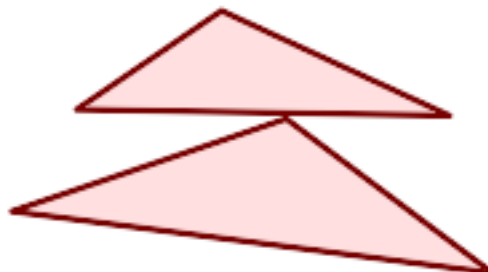
Originals geometries

```
WITH depts_pds as (SELECT ST_GeomFromText('GEOMETRYCOLLECTION(
POLYGON((88.46 158.85,90.77 171.54,147.31 173.85,146.15 145,173.85 119.62,146.15
103.46,112.69 118.46,91.92 93.08,65.38 101.15,34.23 121.92,41.15 142.69,49.23
143.85,88.46 158.85)),
POLYGON((112.69 118.46,146.15 103.46,190 60.77,185.38 43.46,126.54 26.15,83.85 28.46,67.69
64.23,43.46 58.46,10 83.85,34.23 121.92,65.38 101.15,91.92 93.08,112.69 118.46)))
') as geom)
SELECT (ST_Dump(CG_Simplify(geom, 0.5, true))).geom FROM depts_pds;
```



Simplification with 0.5 and topology preserved

```
WITH depts_pds as (SELECT ST_GeomFromText('GEOMETRYCOLLECTION(
POLYGON((88.46 158.85,90.77 171.54,147.31 173.85,146.15 145,173.85 119.62,146.15  ←
103.46,112.69 118.46,91.92 93.08,65.38 101.15,34.23 121.92,41.15 142.69,49.23  ←
143.85,88.46 158.85)),
POLYGON((112.69 118.46,146.15 103.46,190 60.77,185.38 43.46,126.54 26.15,83.85 28.46,67.69  ←
64.23,43.46 58.46,10 83.85,34.23 121.92,65.38 101.15,91.92 93.08,112.69 118.46)))
') as geom)
SELECT (ST_Dump(CG_Simplify(geom, 0.5, false))).geom FROM depts_pds;
```



Simplification with 0.5 without topology preservation

☒☒

[ST_Simplify](#), [ST_SimplifyPreserveTopology](#)

8.3.53 CG_3DAlphaWrapping

CG_3DAlphaWrapping — Computes a 3D Alpha-wrapping strictly enclosing a geometry.

Synopsis

geometry **CG_3DAlphaWrapping**(geometry geom, integer relative_alpha, integer relative_offset);

国国

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



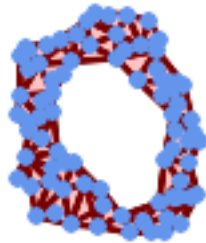
This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.

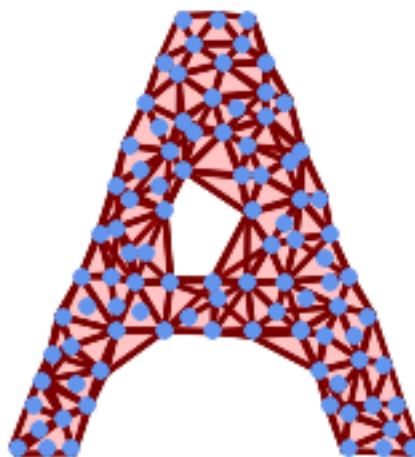
国国

```
SELECT CG_3DAlphaWrapping('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
      (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ←
      ,(36 61),(32 65),
      (81 47),(88 58),(68 73),(49 95),(81 60),(87 50),(78 16),(79 21),(30 22),(78 43) ←
      ,(26 85),(48 34),
      (35 35),(36 40),(31 79),(83 29),(27 84),(52 98),(72 95),(85 71),(75 84),(75 77) ←
      ,(81 29),(77 73),
      (41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77),(39 88),(60 81),(80 72) ←
      ,(54 32),(55 26),
      (62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86),(60 90),(50 86) ←
      ,(43 80),(36 73),
      (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16) ←
      ,(38 46),(31 59),
      (34 86),(45 90),(64 97))'::geometry,10);
```

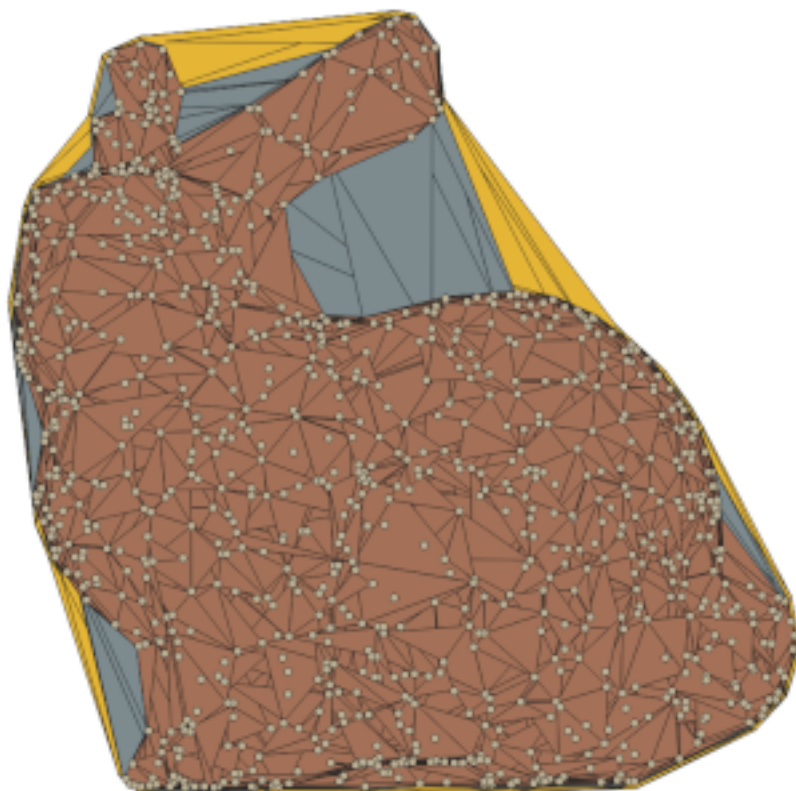


Alpha wrapping of a MultiPoint (same example As [CG_OptimalAlphaShape](#))

```
SELECT CG_3DAlphaWrapping('MULTIPOINT((132 64),(114 64),(99 64),(81 64),(63 64),(57 49),
(52 36),(46 20),(37 20),(26 20),(32 36),(39 55),(43 69),(50 84),(57 100),(63 ←
118),(68 133),(74 149),
(81 164),(88 180),(101 180),(112 180),(119 164),(126 149),(132 131),(139 113) ←
,(143 100),(150 84),(157 69),(163 51),
(168 36),(174 20),(163 20),(150 20),(143 36),(139 49),(132 64),(99 151),(92 138) ←
,(88 124),(81 109),(74 93),(70 82),
(83 82),(99 82),(112 82),(126 82),(121 96),(114 109),(110 122),(103 138),(99 ←
151),(34 27),(43 31),(48 44),(46 58),
(52 73),(63 73),(61 84),(72 71),(90 69),(101 76),(123 71),(141 62),(166 27),(150 ←
33),(159 36),(146 44),(154 53),
(152 62),(146 73),(134 76),(143 82),(141 91),(130 98),(126 104),(132 113),(128 ←
127),(117 122),(112 133),(119 144),
(108 147),(119 153),(110 171),(103 164),(92 171),(86 160),(88 142),(79 140),(72 ←
124),(83 131),(79 118),(68 113),
(63 102),(68 93),(35 45))'::geometry,14);
```



Alpha wrapping of a MultiPoint (same example as [ST_ConcaveHull](#))



Effect of the `relative_alpha` parameter with values 5, 10 and 20. A value of 5 results in a coarse output. Increasing the parameter up to 20 significantly improves the precision and granularity of the result.

☒☒

CG_AlphaShape

Chapter 9

 (topology)

[illegible]

Sandro Santilli's presentation at PostGIS Day Paris 2011 conference gives a good synopsis of PostGIS Topology and where it is headed [Topology with PostGIS 2.0 slide deck](#).

Vincent Picavet provides a good synopsis and overview of what is Topology, how is it used, and various FOSS4G tools that support it in [PostGIS Topology PGConf EU 2012](#).

GIS US Census Topologically Integrated Geographic Encoding and Referencing System (TIGER) PostGIS Topology Load Tiger.

PostGIS 2.0.0 支持 PostGIS 2.0.0 版本, 支持 PostGIS 2.0.0 版本. PostGIS 2.0.0 支持, 支持 PostGIS 2.0.0 版本, 支持 PostGIS 2.0.0 版本. SQL-MM 支持 PostGIS 2.0.0 版本.

PostGIS Topology Wiki.

□□□□□□□□□□□□□□ topology □□□□□□□□□□□□□□.

SQL/MM **ST_**, PostGIS .

Topology support is build by default starting with PostGIS 2.0, and can be disabled specifying `--without-topology` configure option at build time as described in [Chapter 2](#)

9.1 ☐ ☐ ☐ ☐

9.1.1 getfaceedges_returntype

`getfaceedges` `returntype` — A composite type that consists of a sequence number and an edge number.

A composite type that consists of a sequence number and an edge number. This is the return type for `ST_GetFaceEdges` and `GetNodeEdges` functions.

1. sequence `[SRID]`: `[SRID]` SRID `[topology.topology]` `[topology.topology]`.
2. edge `[edge_id]`: `[edge_id]`.

9.1.2 TopoGeometry

TopoGeometry — A composite type representing a topologically defined geometry.

定义

TopoGeometry 由 4 个部分组成: topology_id, layer_id, id, type. TopoGeometry 的每个实例都引用 topology_id, layer_id, id, type 这 4 个部分。

1. topology_id 整数: 表示 SRID 的 topology.topology 的 ID。
2. layer_id 整数: TopoGeometry 所属的 layer_id。 topology_id 和 layer_id 在 topology.layers 表中 (unique reference) 是唯一的。
3. id 整数: 表示 TopoGeometry 的 ID。
4. 1 到 4 之间的 type 值。 1: [点] 点, 2: [线] 线, 3: [面] 面, 4: [体] 体。

结构

TopoGeometry 的 SQL 类型是 geometry。

geometry	geometry
geometry	geometry

函数

CreateTopoGeom

9.1.3 validate_topology_returntype

validate_topology_returntype — A composite type that consists of an error message and id1 and id2 to denote location of error. This is the return type for ValidateTopology.

定义

validate_topology_returntype 由 3 部分组成: error, id1, id2. ValidateTopology 返回 validate_topology_returntype 类型的结果。

1. error 字符串 (varchar) 类型: 描述错误信息。
可能的值 (descriptor) 包括: coincident nodes(重合节点), edge crosses node(边穿过节点), edge not simple(边不简单), edge end node geometry mismatch(边端节点几何不匹配), edge start node geometry mismatch(边起始节点几何不匹配), face overlaps face(面重叠面), face within face(面在面内)。
2. id1 整数: 表示 (边)/面/节点 的 ID。
3. id2 整数: 表示 2 个 ID, 表示面/面/节点。

图元

[ValidateTopology](#)

9.2 图元

9.2.1 TopoElement

TopoElement — 图元是 TopoGeometry 对象的基本组成单元，由 2 个属性组成。

图元

图元是 [TopoGeometry](#) 对象的基本组成单元，由 2 个属性组成。

图元 TopoGeometry 对象，图元是拓扑原语 (topological primitive) 对象的基本组成单元 (1: node, 2: edge, 3: face) 对象。图元 TopoGeometry 对象是图元 TopoGeometry 对象的基本组成单元。



Note

图元 TopoGeometry 对象，图元 TopoGeometry 对象，图元 TopoGeometry 对象是 topology.layer 对象的基本组成单元。

图元

```
SELECT te[1] AS id, te[2] AS type FROM
( SELECT ARRAY[1,2]::topology.topoelement AS te ) f;
id | type
----+-----
1  | 2
```

```
SELECT ARRAY[1,2]::topology.topoelement;
te
-----
{1,2}
```

```
--Example of what happens when you try to case a 3 element array to topoelement
-- NOTE: topoement has to be a 2 element array so fails dimension check
SELECT ARRAY[1,2,3]::topology.topoelement;
ERROR:  value for domain topology.topoelement violates check constraint "dimensions"
```

图元

[GetTopoGeomElements](#), [TopoElementArray](#), [TopoGeometry](#), [TopoGeom_addElement](#), [TopoGeom_remElement](#)

9.2.2 TopoElementArray

TopoElementArray — An array of TopoElement objects.

例

1 创建 TopoElement 表, 创建 TopoElement 表。

例

```
SELECT '{{1,2},{4,3}}'::topology.topoelementarray As tea;
      tea
-----
{{1,2},{4,3}}

-- more verbose equivalent --
SELECT ARRAY[ARRAY[1,2], ARRAY[4,3]]::topology.topoelementarray As tea;
      tea
-----
{{1,2},{4,3}}

--using the array agg function packaged with topology --
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
  FROM generate_series(1,4) As e CROSS JOIN generate_series(1,3) As t;
      tea
-----
{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3},{4,1},{4,2},{4,3}}
```

```
SELECT '{{1,2,4},{3,4,5}}'::topology.topoelementarray As tea;
ERROR:  value for domain topology.topoelementarray violates check constraint "dimensions"
```

例

[TopoElement](#), [GetTopoGeomElementArray](#), [TopoElementArray_Agg](#)

9.3 TopoGeometry 函数

9.3.1 AddTopoGeometryColumn

AddTopoGeometryColumn — 在 TopoGeometry 表中添加 topology.layer 列, 列名为 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);
```

例

1 创建 TopoGeometry 表。 TopoGeometry 表。 AddTopoGeometryColumn() 函数。

topology.layer 函数返回 topology.layer 的元数据。

[child_layer] 如果为 NULL 则返回 NULL，否则返回 (child_layer 的 TopoGeometry 列名) 的 TopoGeometry 列名。如果 child_layer 不为 NULL 则返回 (child_layer 的 TopoGeometry 列名) 的 TopoGeometry 列名。

该函数 (AddTopoGeometryColumn 函数 ID 列) 返回 TopoGeometry 列名。

Valid feature_types are: POINT, MULTIPOINT, LINE, MULTILINE, POLYGON, MULTIPOLYGON, COLLECTION

Availability: 1.1

例

```
-- Note for this example we created our new table in the ma_topo schema
-- though we could have created it in a different schema -- in which case topology_name and ←
-- schema_name would be different
CREATE SCHEMA ma;
CREATE TABLE ma.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('ma_topo', 'ma', 'parcels', 'topo', 'POLYGON');
```

```
CREATE SCHEMA ri;
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);
SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

例

DropTopoGeometryColumn, toTopoGeom, CreateTopology, CreateTopoGeom

9.3.2 RenameTopoGeometryColumn

RenameTopoGeometryColumn — Renames a topogeometry column

Synopsis

topology.layer **RenameTopoGeometryColumn**(regclass layer_table, name feature_column, name new_name)

例

This function changes the name of an existing TopoGeometry column ensuring metadata information about it is updated accordingly.

Availability: 3.4.0

例

```
SELECT topology.RenameTopoGeometryColumn('public.parcels', 'topogeom', 'tgeom');
```

函数

[AddTopoGeometryColumn](#), [RenameTopology](#)

9.3.3 DropTopology

DropTopology — 删除 topology.topology 表中的 topology 和 geometry_columns 表。

Synopsis

```
integer DropTopology(varchar topology_schema_name);
```

函数

删除 topology.topology 表中的 topology 和 geometry_columns 表。该函数将删除 topology 表中的 topology 和 geometry_columns 表。该函数将删除 topology 表中的 topology 和 geometry_columns 表。

Availability: 1.1

函数

ma_topo 删除 topology.topology 表中的 geometry_columns 表。

```
SELECT topology.DropTopology('ma_topo');
```

函数

[DropTopoGeometryColumn](#)

9.3.4 RenameTopology

RenameTopology — 重命名 topology

Synopsis

```
varchar RenameTopology(varchar old_name, varchar new_name);
```

函数

重命名 topology 模式，更新其在 topology.topology 表中的元数据记录。

Availability: 3.4.0

☐☐

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

☐☐

Fix all topology columns

```
SELECT topology.FixCorruptTopoGeometryColumn(schema_name, table_name, feature_column)
FROM topology.layer;
```

☐☐

UpgradeTopology

9.3.7 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 简体中文.

☐☐

```
SELECT CreateTopology('strk_topo');
CREATE SCHEMA strk;
CREATE TABLE strk.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('strk_topo', 'strk', 'parcels', 'topo', 'POLYGON');
-- this will return no records because this feature is already registered
SELECT *
FROM topology.Populate_Topology_Layer();

-- let's rebuild
TRUNCATE TABLE topology.layer;

SELECT *
FROM topology.Populate_Topology_Layer();
```

```
SELECT topology_id, layer_id, schema_name As sn, table_name As tn, feature_column As fc
FROM topology.layer;
```

```
schema_name | table_name | feature_column
-----+-----+-----
strk        | parcels   | topo
(1 row)

topology_id | layer_id | sn  | tn   | fc
-----+-----+-----+-----+-----
          2 |         2 | strk | parcels | topo
(1 row)
```

国国

AddTopoGeometryColumn

9.3.8 TopologySummary

TopologySummary — Takes a topology name and provides summary totals of types of objects in topology.

Synopsis

text **TopologySummary**(varchar topology_schema_name);

国国

Takes a topology name and provides summary totals of types of objects in topology.

2.0.0 国国国国国国国国国国国国.

国国

```
SELECT topology.topologysummary('city_data');
           topologysummary
-----
Topology city_data (329), SRID 4326, precision: 0
22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
Layer 1, type Polygonal (3), 9 topogeoms
  Deploy: features.land_parcel.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_parcel.feature
Layer 5, type Puntal (1), 1 topogeoms
  Hierarchy level 1, child layer 2
  Deploy: features.big_signs.feature
```

函数

[Topology_Load_Tiger](#)

9.3.9 ValidateTopology

ValidateTopology — Returns a set of `validatetopology_returntype` objects detailing issues with topology.

Synopsis

setof `validatetopology_returntype` **ValidateTopology**(varchar toponame, geometry bbox);

函数

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:

函数	id1	id2	Meaning
coincident nodes	Identifier of first node.	Identifier of second node.	Two nodes have the same geometry.
edge crosses node(<code>validatetopology_returntype</code>)	Identifier of the edge.	Identifier of the node.	An edge has a node in its interior. See ST_Relate .
invalid edge(<code>validatetopology_returntype</code>)	Identifier of the edge.		An edge geometry is invalid. See ST_IsValid .
edge not simple(<code>validatetopology_returntype</code>)	Identifier of the edge.		An edge geometry has self-intersections. See ST_IsSimple .
edge crosses edge(<code>validatetopology_returntype</code>)	Identifier of first edge.	Identifier of second edge.	Two edges have an interior intersection. See ST_Relate .
edge start node geometry mismatch(<code>validatetopology_returntype</code>)	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(<code>validatetopology_returntype</code>)	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(<code>validatetopology_returntype</code>)	Identifier of the orphaned face.		No edge reports an existing face on either of its sides (<code>left_face</code> , <code>right_face</code>).

表	id1	id2	Meaning
face has no rings(表)	Identifier of the partially-defined face.		Edges reporting a face on their sides do not form a ring.
face has wrong mbr	Identifier of the face with wrong mbr cache.		Minimum bounding rectangle of a face does not match minimum bounding box of the collection of edges reporting the face on their sides.
hole not in advertised face	Signed identifier of an edge, identifying the ring. See GetRingEdges .		A ring of edges reporting a face on its exterior is contained in different face.
not-isolated node has not- containing_face	Identifier of the ill-defined node.		A node which is reported as being on the boundary of one or more edges is indicating a containing face.
isolated node has containing_face	Identifier of the ill-defined node.		A node which is not reported as being on the boundary of any edges is lacking the indication of a containing face.
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".

	id1	id2	Meaning
non-closed ring	Signed identifier of an edge, identifying the ring. See GetRingEdges .		A ring of edges formed by following <code>next_left_edge/next_right_edge</code> 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.

9.3.11 ValidateTopologyPrecision

ValidateTopologyPrecision — Returns non-precise vertices in the topology.

Synopsis

geometry **ValidateTopologyPrecision**(name toponame, geometry bbox, float8 gridSize);

返回

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

示例

```
SELECT ST_AsEWKT(g) FROM
  topology.ValidateTopologyPrecision(
    'city_data',
    gridSize =
> 2,
    bbox =
> ST_MakeEnvelope(0,0,20,20)
  ) g;
      st_asewkt
-----
MULTIPOINT(9 6,9 14)
(1 row)
```

返回

MakeTopologyPrecise

9.3.12 MakeTopologyPrecise

MakeTopologyPrecise — Snap topology vertices to precision grid.

Synopsis

void **MakeTopologyPrecise**(name toponame, geometry bbox, float8 gridSize);

返回

Snaps all vertices of a topology to the topology precision grid or to the grid whose size is specified with the `gridSize` parameter, optionally limiting the operation to the objects intersecting the area specified by the `bbox` parameter.

**Note**

Snapping could make the topology invalid, so it is recommended to check the outcome of operation with **ValidateTopology**.

Availability: 3.6.0

国国

```
SELECT topology.MakeTopologyPrecise(
    'city_data',
    gridSize =
> 2
);
    maketopologyprecise
-----
(1 row)
```

国国

ValidateTopologyPrecision, ValidateTopology

9.3.13 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);
```

国国

Takes a topology identifier or the identifier of a topology-related object and returns a topology.topology record.

Availability: 3.2.0

国国

```
SELECT name(findTopology('features.land_parcel', 'feature'));
    name
-----
city_data
(1 row)
```

函数

FindLayer

9.3.14 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);
```

函数

Takes a layer identifier or the identifier of a topology-related object and returns a topology.layer record.

Availability: 3.2.0

函数

```
SELECT layer_id(findLayer('features.land_parcel', 'feature'));
 layer_id
-----
        1
(1 row)
```

函数

FindTopology

9.3.15 TotalTopologySize

TotalTopologySize — Total disk space used by the specified topology, including all indexes and TOAST data.

Synopsis

```
int8 TotalTopologySize(name toponame);
```

函数

Takes a topology name and provides the total disk space used by all its tables, including indexes and TOAST data.

Availability: 3.6.0

例例

```
SELECT topology.topologysummary('city_data');
           topologysummary
-----
Topology city_data (329), SRID 4326, precision: 0
22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
Layer 1, type Polygonal (3), 9 topogeoms
  Deploy: features.land_parcels.feature
Layer 2, type Puntal (1), 8 topogeoms
  Deploy: features.traffic_signs.feature
Layer 3, type Lineal (2), 8 topogeoms
  Deploy: features.city_streets.feature
Layer 4, type Polygonal (3), 3 topogeoms
  Hierarchy level 1, child layer 1
  Deploy: features.big_parcels.feature
Layer 5, type Puntal (1), 1 topogeoms
  Hierarchy level 1, child layer 2
  Deploy: features.big_signs.feature
```

例例

Topology_Load_Tiger

9.3.16 UpgradeTopology

UpgradeTopology — Upgrades the specified topology to support large ids (int8) for topology and primitive ids.

Synopsis

void **UpgradeTopology**(name toponame);

例例

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

例例

```
SELECT topology.upgradetopology('city_data');
```

9.4 Topology Statistics Management

Adding elements to a topology triggers many database queries for finding existing edges that will be split, adding nodes and updating edges that will node with the new linework. For this reason it is useful that statistics about the data in the topology tables are up-to-date.

PostGIS Topology population and editing functions do not automatically update the statistics because a updating stats after each and every change in a topology would be overkill, so it is the caller's duty to take care of that.



Note

That the statistics updated by autovacuum will NOT be visible to transactions which started before autovacuum process completed, so long-running transactions will need to run ANALYZE themselves, to use updated statistics.

9.5

9.5.1 CreateTopology

CreateTopology — Creates a new topology schema and registers it in the topology.topology table.

Synopsis

integer **CreateTopology**(name topology_schema_name, integer srid, double precision prec, boolean hasz, integer topoid, boolean useslargeids);



Creates a new topology schema with name `topology_name` and registers it in the `topology.topology` table. Topologies must be uniquely named. The topology tables (`edge_data`, `face`, `node`, and `relation`) are created in the schema. It returns the id of the topology.

The `srid` is the **spatial reference system** SRID for the topology. The SRID defaults to -1 (unknown) if not specified.

The tolerance `prec` is measured in the units of the spatial reference system. The tolerance defaults to 0.

`hasz` defaults to false if not specified.

`topoid` optional explicit identifier (allows deterministic topology id assignment, needs to be unique)

`useslargeids` optional, defaults to false. If true, the topology will be created to support large ids (int8) for topology and primitive ids.

This is similar to the SQL/MM **ST_InitTopoGeo** but has more functionality.

Availability: 1.1

Enhanced: 2.0 added the signature accepting `hasZ`

例例

Create a topology schema called `ma_topo` that stores edges and nodes in Massachusetts State Plane-meters (SRID = 26986). The tolerance represents 0.5 meters since the spatial reference system is meter-based.

```
SELECT topology.CreateTopology('ma_topo', 26986, 0.5);
```

Create a topology for Rhode Island called `ri_topo` in spatial reference system State Plane-feet (SRID = 3438)

```
SELECT topology.CreateTopology('ri_topo', 3438) AS topoid;
topoid
-----
2
```

例例

Section [4.5](#), [ST_InitTopoGeo](#), [Topology_Load_Tiger](#)

9.5.2 CopyTopology

CopyTopology — Makes a copy of a topology (nodes, edges, faces, layers and TopoGeometries) into a new schema

Synopsis

integer **CopyTopology**(varchar existing_topology_name, varchar new_name);

例例

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 例例例例例例例例例例例例例例例例.

例例

Make a backup of a topology called `ma_topo`.

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_backup');
```


例

```
-- Populate topology --
SELECT topology.ST_CreateTopoGeo('ri_topo',
  ST_GeomFromText('MULTILINESTRING((384744 236928,384750 236923,384769 236911,384799 236895,384811 236890,384833 236884,
  384844 236882,384866 236881,384879 236883,384954 236898,385087 236932,385117 236938,
  385167 236938,385203 236941,385224 236946,385233 236950,385241 236956,385254 236971,
  385260 236979,385268 236999,385273 237018,385273 237037,385271 237047,385267 237057,
  385225 237125,385210 237144,385192 237161,385167 237192,385162 237202,385159 237214,
  385159 237227,385162 237241,385166 237256,385196 237324,385209 237345,385234 237375,
  385237 237383,385238 237399,385236 237407,385227 237419,385213 237430,385193 237439,
  385174 237451,385170 237455,385169 237460,385171 237475,385181 237503,385190 237521,
  385200 237533,385206 237538,385213 237541,385221 237542,385235 237540,385242 237541,
  385249 237544,385260 237555,385270 237570,385289 237584,385292 237589,385291 237596,385284 237630))',3438)
);

      st_createtopogeo
-----
Topology ri_topo populated

-- create tables and topo geometries --
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);

SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

例

[TopoGeo_LoadGeometry](#), [AddTopoGeometryColumn](#), [CreateTopology](#), [DropTopology](#)

9.5.5 TopoGeo_AddPoint

`TopoGeo_AddPoint` — 向拓扑中添加点并返回其标识符 (split) 函数

Synopsis

`bigint TopoGeo_AddPoint`(varchar atopoology, geometry apoint, float8 tolerance);

例

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 新增。

例

[TopoGeo_AddLineString](#), [TopoGeo_AddPolygon](#), [TopoGeo_LoadGeometry](#), [AddNode](#), [CreateTopology](#)

9.5.6 TopoGeo_AddLineString

TopoGeo_AddLineString — Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.

Synopsis

SETOF bigint **TopoGeo_AddLineString**(varchar atopolgy, geometry aline, float8 tolerance);

国国

Adds a linestring to an existing topology and returns a set of signed edge identifiers forming it up (negative identifies mean the edge goes in the opposite direction of the input linestring). The given line will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the line. New nodes and faces may be added.



Note

Updating statistics about topologies being loaded via this function is up to caller, see [maintaining statistics during topology editing and population](#).

2.0.0 国国国国国国国国国国国国.

Enhanced: 3.2.0 added support for returning signed identifier.

国国

[TopoGeo_AddPoint](#), [TopoGeo_AddPolygon](#), [TopoGeo_LoadGeometry](#), [AddEdge](#), [CreateTopology](#)

9.5.7 TopoGeo_AddPolygon

TopoGeo_AddPolygon — Adds a polygon to an existing topology using a tolerance and possibly splitting existing edges/faces. Returns face identifiers.

Synopsis

SETOF bigint **TopoGeo_AddPolygon**(varchar atopolgy, geometry apoly, float8 tolerance);

国国

Adds a polygon to an existing topology and returns a set of face identifiers forming it up. The boundary of the given polygon will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the boundary of the new polygon.



Note

Updating statistics about topologies being loaded via this function is up to caller, see [maintaining statistics during topology editing and population](#).

2.0.0 国国国国国国国国国国国国.

函数

[TopoGeo_AddPoint](#), [TopoGeo_AddLineString](#), [TopoGeo_LoadGeometry](#), [AddFace](#), [CreateTopology](#)

9.5.8 TopoGeo_LoadGeometry

`TopoGeo_LoadGeometry` — Load a geometry into an existing topology, snapping and splitting as needed.

Synopsis

```
void TopoGeo_LoadGeometry(varchar atopology, geometry ageom, float8 tolerance);
```

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](#)

9.6 函数

9.6.1 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 atopology, bigint aface, geometry apoint);
```



```
atopology xxx aface ID(faceid) xxxxxx apoint xxxxxxxx  
ID(nodeid) xxxxx.
```

SRID (SRID) 指定, apoint 指定, NULL 指定, 指定 (指定) 指定. 指定 指定.

```
aface = NULL; apoint[0] = 0;
```

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X+1.3.1

AddNode, CreateTopology, DropTopology, ST_Intersects

9.6.2 ST_AddIsoEdge

```
ST_AddIsoEdge — void ST_AddIsoEdge(ST_GeometryContext anode ST_GeometryContext anothernode const char* alinestring const int ID)
```

Synopsis

bigint **ST_AddIsoEdge**(varchar atopology, bigint anode, bigint anothernode, geometry alinestring);

```

graph LR
    anode --> anothernode
    anode --> alinestring
    ID(edgeid)

```

```
aliningstring [padding] (SRID) [precision], [scale] NULL [mode], [mode]
[padding] [padding], [padding] [padding], [padding].
```

```
alinestring  x anode  x anothernode  xxxxxxxxxxxxxxxxxxxxxxxx, xxxxxxxxxxxx.
```

```
anode □ anothernode □ alinestring □□□□□□□□□□□□□□□□□□.
```

Availability: 1.1



✔ This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4



ST AddIsoNode, ST IsSimple, ST Within

9.6.3 ST_AddEdgeNewFaces

`ST_AddEdgeNewFaces` — 添加新面, 添加新面, 添加新面
返回 2 个面。

Synopsis

`bigint ST_AddEdgeNewFaces`(varchar atopology, bigint anode, bigint anothernode, geometry acurve);

返回

返回面 ID, 添加新面, 添加新面, 添加新面 2 个面。返回面 ID 列表。

添加新面, 添加新面, 添加新面, 添加新面, 添加新面。

返回 NULL 面, 添加新面, 添加新面 (添加新面 node 面), acurve 为 LINESTRING 面, anode 为 anothernode 面 acurve 面。

acurve 面 (SRID) 面。

2.0 面。



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`

9.6.4 ST_AddEdgeModFace

`ST_AddEdgeModFace` — 添加新面, 添加新面, 添加新面
返回面 ID。

Synopsis

`bigint ST_AddEdgeModFace`(varchar atopology, bigint anode, bigint anothernode, geometry acurve);

返回

添加新面, 添加新面, 添加新面, 添加新面, 添加新面。



Note

添加新面, 添加新面, 添加新面, 添加新面, 添加新面。添加新面 (面) 面 (universe face) 面。

返回的几何体的 ID 值。

返回的几何体的 SRID 值。

如果 NULL 值, 则返回的几何体 (即返回的 node 值), acurve 为 LINESTRING 类型, anode 为 anothernode 为 acurve 的起始点。

acurve 的 SRID 值 (SRID) 值。

2.0 版本。



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13

ST

ST

[ST_RemEdgeModFace](#)

[ST_AddEdgeNewFaces](#)

9.6.5 ST_RemEdgeNewFace

ST_RemEdgeNewFace — 移除拓扑中的边, 并创建新的面, 返回新的拓扑。

Synopsis

bigint **ST_RemEdgeNewFace**(varchar atopology, bigint anedge);

ST

返回的拓扑, 返回的拓扑的 SRID 值, 返回的拓扑的 SRID 值。

返回的 ID 值, 返回的 NULL 值。返回的拓扑, 返回的拓扑, 返回的拓扑 (即返回的 edge 值) 返回的拓扑。

返回的 SRID 值。

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 NULL 值, 则返回的几何体 (即返回的 edge 值), 返回的拓扑, 返回的拓扑。

2.0 版本。



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14

ST

ST

[ST_RemEdgeModFace](#)

[ST_AddEdgeNewFaces](#)

9.6.6 ST_RemEdgeModFace

ST_RemEdgeModFace — Removes an edge, and if the edge separates two faces deletes one face and modifies the other face to cover the space of both.

Synopsis

```
bigint ST_RemEdgeModFace(varchar atopology, bigint anedge);
```


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

9.6.7 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 ████████████████████ (SRID) ████████████████████.

`acurve` 返回曲线，如果存在。

返回的曲线是唯一的，并且是闭合的。

1.1.0 版本引入。

更新: 2.0.0 版本引入。



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details X.3.6

SQL

```
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
```

SQL

`ST_AddEdgeModFace`

`ST_RemEdgeModFace`

`ST_ModEdgeSplit`

9.6.8 ST_ModEdgeSplit

`ST_ModEdgeSplit` — 修改拓扑中的边，使其在指定点处分裂。

Synopsis

`bigint ST_ModEdgeSplit(varchar atopology, bigint anedge, geometry apoint);`

SQL

返回的边是唯一的，并且是闭合的。如果点位于边上，则返回的边是唯一的，并且是闭合的。如果点不在边上，则返回的边是唯一的，并且是闭合的。

Availability: 1.1

更新: 2.0 版本引入，同时 `ST_ModEdgesSplit` 也被引入。



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

❏

```
-- Add an edge --
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227592 893910, 227600 893910)', 26986) ) As edgeid;

-- edgeid-
3

-- Split the edge --
SELECT topology.ST_ModEdgeSplit('ma_topo', 3, ST_SetSRID(ST_Point(227594,893910),26986) ) As node_id;
       node_id
-----
7
```

❏

[ST_NewEdgesSplit](#), [ST_ModEdgeHeal](#), [ST_NewEdgeHeal](#), [AddEdge](#)

9.6.9 ST_ModEdgeHeal

ST_ModEdgeHeal — Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node.

Synopsis

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 简体中文.



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

❏

[ST_ModEdgeSplit](#) [ST_NewEdgesSplit](#)

9.6.10 ST_NewEdgeHeal

ST_NewEdgeHeal — Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided.

Synopsis

bigint **ST_NewEdgeHeal**(varchar atopology, bigint anedge, bigint anotheredge);

国国

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 国国国国国国国国国国国国.

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

国国

ST_ModEdgeHeal **ST_ModEdgeSplit** **ST_NewEdgesSplit**

9.6.11 ST_MoveIsoNode

ST_MoveIsoNode — Moves an isolated node in a topology from one point to another. If new apoint geometry exists as a node an error is thrown. Returns description of move.

Synopsis

text **ST_MoveIsoNode**(varchar atopology, bigint anode, geometry apoint);

国国

国国国国国国国国国国国国国国国国国国国国国国国国. 国 apoint 国国国国国国国国国国国国国国国国国国国国国国国国.

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.

国国国国国国国国国国国国国国国国 (SRID) 国国国国国国国国国国国国.

2.0.0 国国国国国国国国国国国国.

Enhanced: 3.2.0 ensures the nod cannot be moved in a different face

 This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.2

国国

```
-- Add an isolated node with no face --
SELECT topology.ST_AddIsoNode('ma_topo', NULL, ST_GeomFromText('POINT(227579 893916)', 26986) ) As nodeid;
nodeid
-----
7
-- Move the new node --
```

```
SELECT topology.ST_MoveIsoNode('ma_topo', 7, ST_GeomFromText('POINT(227579.5 893916.5)', ←
    26986) ) As descrip;
        descrip
-----
Isolated Node 7 moved to location 227579.5,893916.5
```

☐☐

ST_AddIsoNode

9.6.12 ST_NewEdgesSplit

ST_NewEdgesSplit — 将边分割成两部分，并返回新边的 ID。将边分割成两部分，并返回新边的 ID。将边分割成两部分，并返回新边的 ID。

Synopsis

bigint **ST_NewEdgesSplit**(varchar atopology, bigint anedge, geometry apoint);

☐☐

将边分割成两部分，并返回新边的 ID。将边分割成两部分，并返回新边的 ID。将边分割成两部分，并返回新边的 ID。将边分割成两部分，并返回新边的 ID。将边分割成两部分，并返回新边的 ID。

将边分割成两部分，并返回新边的 ID。将边分割成两部分，并返回新边的 ID。将边分割成两部分，并返回新边的 ID。将边分割成两部分，并返回新边的 ID。将边分割成两部分，并返回新边的 ID。

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.8

☐☐

```
-- Add an edge --
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575 893917,227592 893900)' ←
    ', 26986) ) As edgeid;
-- result-
edgeid
-----
      2
-- Split the new edge --
SELECT topology.ST_NewEdgesSplit('ma_topo', 2, ST_GeomFromText('POINT(227578.5 893913.5)', ←
    26986) ) As newnodeid;
    newnodeid
-----
      6
```

☐☐

ST_ModEdgeSplit ST_ModEdgeHeal ST_NewEdgeHeal AddEdge

9.6.13 ST_RemoveIsoNode

ST_RemoveIsoNode — 移除拓扑中的孤立节点并返回描述。如果节点不是孤立的，则抛出异常。

Synopsis

text **ST_RemoveIsoNode**(varchar atopology, bigint anode);

返回

移除拓扑中的孤立节点并返回描述。如果节点不是孤立的，则抛出异常。

Availability: 1.1

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

示例

```
-- Remove an isolated node with no face --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7 ) As result;
-----
Isolated node 7 removed
```

返回

ST_AddIsoNode

9.6.14 ST_RemoveIsoEdge

ST_RemoveIsoEdge — 移除拓扑中的孤立边并返回描述。如果边不是孤立的，则抛出异常。

Synopsis

text **ST_RemoveIsoEdge**(varchar atopology, bigint anedge);

返回

移除拓扑中的孤立边并返回描述。如果边不是孤立的，则抛出异常。

Availability: 1.1

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

❏

```
-- Remove an isolated node with no face --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7 ) As result;
           result
-----
Isolated node 7 removed
```

❏

ST_AddIsoNode

9.7 几何函数

9.7.1 GetEdgeByPoint

GetEdgeByPoint — Finds the edge-id of an edge that intersects a given point.

Synopsis

bigint **GetEdgeByPoint**(varchar atopology, geometry apoint, float8 tol1);

❏

Retrieves the id of an edge that intersects a Point.

❏, ❏, ❏ (edgeid) ❏. tolerance = 0 ❏.

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

❏ tolerance = 0 ❏ ST_Intersects ❏, ❏ ST_DWithin ❏.

GEOS ❷

2.0.0 ❷.

❏

❏ **AddEdge** ❏.

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As with1mtol, topology.GetEdgeByPoint(' ↵
ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('SRID=26986;POINT(227622.6 893843)') As geom;
with1mtol | withnotol
-----+-----
2 | 0
```

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

-- get error --
ERROR:  Two or more edges found
```

☐☐

[AddEdge](#), [GetNodeByPoint](#), [GetFaceByPoint](#)

9.7.2 GetFaceByPoint

GetFaceByPoint — Finds face intersecting a given point.

Synopsis

bigint **GetFaceByPoint**(varchar atopology, geometry apoint, float8 tol1);

☐☐

Finds a face referenced by a Point, with given tolerance.

The function will effectively look for a face intersecting a circle having the point as center and the tolerance as radius.

If no face intersects the given query location, 0 is returned (universal face).

If more than one face intersect the query location an exception is thrown.

2.0.0 ☐☐☐☐☐☐☐☐☐☐.

Enhanced: 3.2.0 more efficient implementation and clearer contract, stops working with invalid topologies.

☐☐

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 10) As with1mtol, topology.GetFaceByPoint('ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('POINT(234604.6 899382.0)') As geom;

with1mtol | withnotol
-----+-----
1 | 0
```

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('POINT(227591.9 893900.4)') As geom;

-- get error --
ERROR:  Two or more faces found
```

☐☐

[GetFaceContainingPoint](#), [AddFace](#), [GetNodeByPoint](#), [GetEdgeByPoint](#)

9.7.3 GetFaceContainingPoint

GetFaceContainingPoint — Finds the face containing a point.

Synopsis

bigint **GetFaceContainingPoint**(text atopology, geometry apoint);

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

国国

[ST_GetFaceGeometry](#)

9.7.4 GetNodeByPoint

GetNodeByPoint — Finds the node-id of a node at a point location.

Synopsis

bigint **GetNodeByPoint**(varchar atopology, geometry apoint, float8 tol1);

国国

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

国国国国 tolerance = 0 国国 ST_Intersects 国, 国国国国国国 ST_DWithin 国国国国国国.

GEOS 国国国国国

2.0.0 国国国国国国国国国国国.

例

使用 `AddEdge` 添加边。

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
nearnode
-----
2
```

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1000) As too_much_tolerance
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

---get error--
ERROR:  Two or more nodes found
```

例

`AddEdge`, `GetEdgeByPoint`, `GetFaceByPoint`

9.7.5 GetTopologyID

`GetTopologyID` — 返回 topology.topology 的 ID。

Synopsis

integer **GetTopologyID**(varchar toponame);

例

返回 topology.topology 的 ID。

Availability: 1.1

例

```
SELECT topology.GetTopologyID('ma_topo') As topo_id;
topo_id
-----
1
```

例

`CreateTopology`, `DropTopology`, `GetTopologyName`, `GetTopologySRID`

9.7.6 GetTopologySRID

`GetTopologySRID` — 返回 topology.topology 的 SRID。

Synopsis

integer **GetTopologyID**(varchar toponame);

返回

返回 topology.topology 的 ID。从 2.0.0 版本开始。

示例

```
SELECT topology.GetTopologySRID('ma_topo') As SRID;
SRID
-----
4326
```

参见

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologyID](#)

9.7.7 GetTopologyName

GetTopologyName — 返回 ID 为 (id) 的 topology 名称。

Synopsis

varchar **GetTopologyName**(integer topology_id);

返回

返回 ID 为 topology.topology (id) 的名称。
Availability: 1.1

示例

```
SELECT topology.GetTopologyName(1) As topo_name;
topo_name
-----
ma_topo
```

参见

[CreateTopology](#), [DropTopology](#), [GetTopologyID](#), [GetTopologySRID](#)

9.7.8 ST_GetFaceEdges

ST_GetFaceEdges — aface 的边界边。

Synopsis

```
getfaceedges_returntype ST_GetFaceEdges(varchar atopology, bigint aface);
```

参数

aface 面 ID。 返回的边按 (sequence) 升序排列 ID(edgeid) 升序排列。 返回 1 条边。

返回的边按 (sequence) 升序排列 ID(edgeid) 升序排列。 返回 1 条边。

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

示例

```
-- Returns the edges bounding face 1
SELECT (topology.ST_GetFaceEdges('tt', 1)).*;
-- result --
sequence | edge
-----+-----
1 | -4
2 | 5
3 | 7
4 | -6
5 | 1
6 | 2
7 | 3
(7 rows)
```

```
-- Returns the sequence, edge id
-- and geometry of the edges that bound face 1
-- If you just need geom and seq, can use ST_GetFaceGeometry
SELECT t.seq, t.edge, geom
FROM topology.ST_GetFaceEdges('tt',1) As t(seq,edge)
INNER JOIN tt.edge AS e ON abs(t.edge) = e.edge_id;
```

相关链接

[GetRingEdges](#), [AddFace](#), [ST_GetFaceGeometry](#)

9.7.9 ST_GetFaceGeometry

ST_GetFaceGeometry — 面 ID 的几何。

Synopsis

geometry **ST_GetFaceGeometry**(varchar atopology, bigint aface);

返回

返回指定拓扑 ID 的面的几何体。返回的几何体是面几何体。

Availability: 1.1

 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.16

返回

```
-- Returns the wkt of the polygon added with AddFace
SELECT ST_AsText(topology.ST_GetFaceGeometry('ma_topo', 1)) As facegeomwkt;
-- result --
          facegeomwkt
-----
POLYGON((234776.9 899563.7,234896.5 899456.7,234914 899436.4,234946.6 899356.9,
234872.5 899328.7,234891 899285.4,234992.5 899145,234890.6 899069,
234755.2 899255.4,234612.7 899379.4,234776.9 899563.7))
```

返回

AddFace

9.7.10 GetRingEdges

GetRingEdges — 返回指定拓扑 ID 的面的环的几何体。

Synopsis

getfaceedges_returntype **GetRingEdges**(varchar atopology, bigint aring, integer max_edges=null);

返回

返回指定拓扑 ID 的面的环的几何体。返回的几何体是面几何体 (sequence) 的 ID(edgeid) 的列表。返回 1 个列表。

返回指定拓扑 ID 的面的环的几何体，返回的几何体是面几何体。返回指定 ID 的面的环的几何体，返回的几何体是面几何体。

max_edges 为 NULL 时返回指定拓扑 ID 的面的环的几何体。返回指定 ID 的面的环的几何体。



Note

返回指定拓扑 ID 的面的环的几何体。

2.0.0 返回指定拓扑 ID 的面的环的几何体。

函数

[ST_GetFaceEdges](#), [GetNodeEdges](#)

9.7.11 GetNodeEdges

GetNodeEdges — 返回拓扑中指定节点的所有边。

Synopsis

```
getfaceedges_returntype GetNodeEdges(varchar atopology, bigint anode);
```

函数

返回指定拓扑 ID 的指定节点的所有边。节点 ID 从 1 开始。返回的边是拓扑中的边 ID。返回的边 ID 是 (节点 ID, 边 ID) 的元组。返回的边 ID 是 (节点 ID, 边 ID) 的元组。



Note

返回的边 ID 是 (节点 ID, 边 ID) 的元组。

2.0 版本引入。

函数

[getfaceedges_returntype](#), [GetRingEdges](#), [ST_Azimuth](#)

9.8 多边形

9.8.1 Polygonize

Polygonize — Finds and registers all faces defined by topology edges.

Synopsis

```
text Polygonize(varchar toponame);
```

函数

Registers all faces that can be built out a topology edge primitives.

返回的边 ID 是 (节点 ID, 边 ID) 的元组。

**Note**

如果面已经存在，`Polygonize` 将返回空结果集。

**Note**

每个 `edge` 都有 `next_left_edge` 和 `next_right_edge` 属性。

2.0.0 版本引入。

参见

[AddFace](#), [ST_Polygonize](#)

9.8.2 AddNode

`AddNode` — 在面中添加一个新节点。指定 `ID(nodeid)` 标识新节点。如果面已经存在，返回 `ID` 值。

Synopsis

`bigint AddNode(varchar toponame, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);`

参见

如果面已经存在，`AddEdge` 将返回空结果集。如果面不存在，返回 `ID` 值。

如果面已经存在，`allowEdgeSplitting` 属性将返回空结果集。

`computeContainingFace` 属性将返回空结果集。

**Note**

`apoint` 指定新节点的位置，返回 `ID(nodeid)` 值。

2.0.0 版本引入。

参见

```
SELECT topology.AddNode('ma_topo', ST_GeomFromText('POINT(227641.6 893816.5)', 26986) ) As
    nodeid;
-- result --
nodeid
-----
4
```

图

AddEdge, CreateTopology

9.8.3 AddEdge

AddEdge — 向拓扑中添加边，并返回边的ID。该函数接受一个拓扑名称、一个几何体（LINESTRING）以及一个可选的ID（edgeid）。

Synopsis

```
bigint AddEdge(varchar toponame, geometry aline);
```

图

该函数接受一个拓扑名称 **toponame**，一个几何体 **aline**（LINESTRING），以及一个可选的ID（edgeid）。该函数返回一个整数，表示添加的边的ID。该函数在“宇宙”（universe）中操作。



Note

aline 必须是一个有效的 LINESTRING 几何体。如果 **aline** 不是一个有效的 LINESTRING，函数将返回 NULL。



Note

aline 的 SRID 必须与拓扑的 SRID 相同。如果 **aline** 的 SRID 与拓扑的 SRID 不同，函数将返回 NULL。

GEOS 支持



Warning

AddEdge 已弃用，自 3.5.0 版本起。请使用 **TopoGeo_AddLineString** 代替。

2.0.0 版本支持。

图

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575.8 893917.2,227591.9
  893900.4)', 26986) ) As edgeid;
-- result-
edgeid
-----
1

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6
  893844.2,227641.6 893816.5,
```

```
227704.5 893778.5)', 26986) ) As edgeid;
-- result --
edgeid
-----
2

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.2 893900, 227591.9 893900.4, 227704.5 893778.5)', 26986) ) As edgeid;
-- gives error --
ERROR:  Edge intersects (not on endpoints) with existing edge 1
```



TopoGeo AddLineString, CreateTopology, Section 4.5

9.8.4 AddFace

AddFace — □□□□□□□ (face primitive) □□□□□□□□□□□□□□.

Synopsis

```
bigint AddFace(varchar toponame, geometry apolygon, boolean force new=false);
```



 (face primitive) .

`left_face` `right_face` containing_face



Note!

Note

edge next left edge next right edge.

本報告係根據本會所屬之「國家安全情報中心」(以下簡稱「情報中心」)提供之資料彙編而成。本報告所載之資料，係根據本會所屬之「國家安全情報中心」(以下簡稱「情報中心」)提供之資料彙編而成。本報告所載之資料，係根據本會所屬之「國家安全情報中心」(以下簡稱「情報中心」)提供之資料彙編而成。

apolygon 0000000000000000, force_new 0 (0000) 0000000000 ID 0000
0, force new 0000000000000000 ID 00000000.



Note

1. `force_new = true` 强制新建，覆盖原有文件。



Note

apolygon [hexagon] srid [hexagon] srid [hexagon]. [hexagon]
[hexagon].

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

❏

```
-- first add the edges we use generate_series as an iterator (the below
-- will only work for polygons with < 10000 points because of our max in gs)
SELECT topology.AddEdge('ma_topo', ST_MakeLine(ST_PointN(geom,i), ST_PointN(geom, i + 1) )) ↵
    As edgeid
FROM (SELECT ST_NPoints(geom) AS npt, geom
      FROM
        (SELECT ST_Boundary(ST_GeomFromText('POLYGON((234896.5 899456.7,234914 ↵
          899436.4,234946.6 899356.9,234872.5 899328.7,
          234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
          234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) ) As geom
        ) As geoms) As facen CROSS JOIN generate_series(1,10000) As i
      WHERE i < npt;
-- result --
edgeid
-----
      3
      4
      5
      6
      7
      8
      9
     10
     11
     12
(10 rows)
-- then add the face -

SELECT topology.AddFace('ma_topo',
    ST_GeomFromText('POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5 ↵
      899328.7,
      234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
      234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) ) As faceid;
-- result --
faceid
-----
      1
```

❏

[AddEdge](#), [CreateTopology](#), [Section 4.5](#)

9.8.5 ST_Simplify

ST_Simplify — ❏❏❏❏-❏❏ (Douglas-Peucker) ❏❏❏❏❏❏❏❏ TopoGeometry ❏” ❏❏❏❏” ❏❏❏❏❏❏❏❏.

Synopsis

geometry **ST_Simplify**(topogeometry tg, float8 tolerance);

图

图 (Douglas-Peucker) 图 TopoGeometry 图 图



Note

图, 图.
图/图.

GEOS 图

2.1.0 图.

图

图 [ST_Simplify](#), [ST_IsSimple](#), [ST_IsValid](#), [ST_ModEdgeSplit](#)

9.8.6 RemoveUnusedPrimitives

RemoveUnusedPrimitives — Removes topology primitives which not needed to define existing TopoGeometry objects.

Synopsis

bigint **RemoveUnusedPrimitives**(text topology_name, geometry bbox);

图

Finds all primitives (nodes, edges, faces) that are not strictly needed to represent existing TopoGeometry objects and removes them, maintaining topology validity (edge linking, face labeling) and TopoGeometry space occupation.

No new primitive identifiers are created, but rather existing primitives are expanded to include merged faces (upon removing edges) or healed edges (upon removing nodes).

Availability: 3.3.0

图

[ST_ModEdgeHeal](#), [ST_RemEdgeModFace](#)

9.9 TopoGeometry 图

9.9.1 CreateTopoGeom

CreateTopoGeom — 图. tg_type 图 1: [图] 图, 2: [图] 图, 3: [图] 图, 4: 图.


```
--the world is rarely perfect allow for some error
--count the face if 50% of it falls
-- within what we think is our blockgroup boundary
UPDATE boston.blockgroups AS bg
    SET topo = topology.CreateTopoGeom('topo_boston'
    ,3,1
    , foo.bfaces)
FROM (SELECT b.gid, topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
    FROM boston.blockgroups As b
    INNER JOIN topo_boston.face As f ON b.geom && f.mbr
    WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
    OR
    ( ST_Intersects(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
    AND ST_Area(ST_Intersection(b.geom, topology.ST_GetFaceGeometry('topo_boston',
    f.face_id) ) ) >
    ST_Area(topology.ST_GetFaceGeometry('topo_boston', f.face_id))*0.5
    )
    GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;

-- and if we wanted to convert our topogeometry back
-- to a denormalized geometry aligned with our faces and edges
-- cast the topo to a geometry
-- The really cool thing is my new geometries
-- are now aligned with my tiger street centerlines
UPDATE boston.blockgroups SET new geom = topo::geometry;
```



AddTopoGeometryColumn, toTopoGeom ST_CreateTopoGeo, ST_GetFaceGeometry, TopoElementArray, TopoElementArray Agg

9.9.2 toTopoGeom

toTopoGeom — Converts a simple Geometry into a topo geometry.

Synopsis

```
topogeometry toTopoGeom(geometry geom, varchar toponame, integer layer_id, float8 tolerance);
topogeometry toTopoGeom(geometry geom, topogeometry topogeom, float8 tolerance);
```


TopoGeometry.

```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.
XXXXXXXXXX relation XXXXXXXXXXXXX TopoGeometry XXXXXXXXXXXXX.

```

TopoGeometry (topogeom) [TopoGeometry \(topogeom\)](#)

tolerance ██.

1 (toponame) (layer id) TopoGeometry.

第 2 步, 调用 TopoGeometry(toponame) 初始化 TopoGeometry 对象。调用 TopoGeometry 初始化 TopoGeometry 对象。调用 clearTopoGeom 清除 TopoGeometry 对象。

2.0 创建拓扑。

步骤: 2.1.0 创建 TopoGeometry 表。

步骤

创建拓扑 (workflow) 步骤。

```
-- do this if you don't have a topology setup already
-- creates topology not allowing any tolerance
SELECT topology.CreateTopology('topo_boston_test', 2249);
-- create a new table
CREATE TABLE nei_topo(gid serial primary key, nei varchar(30));
--add a topogeometry column to it
SELECT topology.AddTopoGeometryColumn('topo_boston_test', 'public', 'nei_topo', 'topo', 'MULTIPOLYGON') As new_layer_id;
new_layer_id
-----
1

--use new layer id in populating the new topogeometry column
-- we add the topogeoms to the new layer with 0 tolerance
INSERT INTO nei_topo(nei, topo)
SELECT nei, topology.toTopoGeom(geom, 'topo_boston_test', 1)
FROM neighborhoods
WHERE gid BETWEEN 1 and 15;

--use to verify what has happened --
SELECT * FROM
    topology.TopologySummary('topo_boston_test');

-- summary--
Topology topo_boston_test (5), SRID 2249, precision 0
61 nodes, 87 edges, 35 faces, 15 topogeoms in 1 layers
Layer 1, type Polygonal (3), 15 topogeoms
Deploy: public.nei_topo.topo

-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = toTopoGeom(ST_Buffer(topo, -10), clearTopoGeom(topo), 0);

-- Get the no-one-lands left by the above operation
-- I think GRASS calls this "polygon0 layer"
SELECT ST_GetFaceGeometry('topo_boston_test', f.face_id)
FROM topo_boston_test.face f
WHERE f.face_id
> 0 -- don't consider the universe face
AND NOT EXISTS ( -- check that no TopoGeometry references the face
    SELECT * FROM topo_boston_test.relation
    WHERE layer_id = 1 AND element_id = f.face_id
);
```

步骤

CreateTopology, AddTopoGeometryColumn, CreateTopoGeom, TopologySummary, clearTopoGeom

9.9.3 TopoElementArray_Agg

TopoElementArray_Agg — Returns a **topoelementarray** for a set of **element_id**, type arrays (**topoelements**).

Synopsis

topoelementarray **TopoElementArray_Agg**(**topoelement set** **tefield**);

例

TopoElement 函数返回 **TopoElementArray** 函数返回的结果。

2.0.0 版本引入。

例

```
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
FROM generate_series(1,3) As e CROSS JOIN generate_series(1,4) As t;
tea
-----
{{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4}}
```

例

TopoElement, **TopoElementArray**

9.9.4 TopoElement

TopoElement — Converts a **topogeometry** to a **topoelement**.

Synopsis

topoelement **TopoElement**(**topogeometry** **topo**);

例

Converts a **TopoGeometry** to a **TopoElement**.

Availability: 3.4.0

例

以下函数在拓扑设置中 (workflow) 使用。

```
-- do this if you don't have a topology setup already
-- Creates topology not allowing any tolerance
SELECT TopoElement(topo)
FROM neighborhoods;
```

```
-- using as cast
SELECT topology.TopoElementArray_Agg(topo::topoelement)
FROM neighborhoods
GROUP BY city;
```

関数

[TopoElementArray_Agg](#), [TopoGeometry](#), [TopoElement](#)

9.10 TopoGeometry 関数

9.10.1 clearTopoGeom

`clearTopoGeom` — Clears the content of a topo geometry.

Synopsis

`topogeometry` **clearTopoGeom**(`topogeometry` `topogeom`);

関数

[TopoGeometry](#) 関数は [TopoGeometry](#) オブジェクトを [toTopoGeom](#) 関数に変換します。

2.1 関数。

関数

```
-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = toTopoGeom(ST_Buffer(topo, -10), clearTopoGeom(topo), 0);
```

関数

[toTopoGeom](#)

9.10.2 TopoGeom_addElement

`TopoGeom_addElement` — Adds an element to the definition of a TopoGeometry.

Synopsis

`topogeometry` **TopoGeom_addElement**(`topogeometry` `tg`, `topoelement` `el`);

图元

TopoGeometry 函数返回 **TopoElement** 图元。图元是 TopoGeometry 的组成部分。

2.3 图元。

图元

```
-- Add edge 5 to TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_addElement(tg, '{5,2}');
```

图元

TopoGeom_remElement, CreateTopoGeom

9.10.3 TopoGeom_remElement

TopoGeom_remElement — Removes an element from the definition of a TopoGeometry.

Synopsis

topogeometry **TopoGeom_remElement**(topogeometry tg, topoelement el);

图元

TopoGeometry 函数返回 **TopoElement** 图元。

2.3 图元。

图元

```
-- Remove face 43 from TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_remElement(tg, '{43,3}');
```

图元

TopoGeom_addElement, CreateTopoGeom

9.10.4 TopoGeom_addTopoGeom

TopoGeom_addTopoGeom — Adds element of a TopoGeometry to the definition of another TopoGeometry.

Synopsis

topogeometry **TopoGeom_addTopoGeom**(topogeometry tgt, topogeometry src);

函数

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

函数

```
-- Set an "overall" TopoGeometry value to be composed by all
-- elements of specific TopoGeometry values
UPDATE mylayer SET tg_overall = TopoGeom_addTopogeom(
    TopoGeom_addTopoGeom(
        clearTopoGeom(tg_overall),
        tg_specific1
    ),
    tg_specific2
);
```

函数

TopoGeom_addElement, **clearTopoGeom**, **CreateTopoGeom**

9.10.5 toTopoGeom

toTopoGeom — Adds a geometry shape to an existing topo geometry.

函数

Refer to **toTopoGeom**.

9.11 TopoGeometry 函数

9.11.1 GetTopoGeomElementArray

GetTopoGeomElementArray — Returns a `topoelementarray` (an array of `topoelements`) containing the topological elements and type of the given TopoGeometry (primitive elements).

Synopsis

```
topoelementarray GetTopoGeomElementArray(varchar toponame, integer layer_id, bigint tg_id);
topoelementarray GetTopoGeomElementArray(topogeometry tg);
```

返回

返回 `TopoGeometry` 对象 (对象) 的 `TopoElementArray`。 使用 `GetTopoGeomElements` 函数。

`tg_id` 是 `topology.layer` 的 `layer_id` 的 `TopoGeometry` 的 `TopoGeometry ID`。

Availability: 1.1

返回

返回

`GetTopoGeomElements`, `TopoElementArray`

9.11.2 GetTopoGeomElements

`GetTopoGeomElements` — Returns a set of `topoelement` objects containing the topological `element_id`, `element_type` of the given `TopoGeometry` (primitive elements).

Synopsis

setof `topoelement` **GetTopoGeomElements**(varchar `toponame`, integer `layer_id`, bigint `tg_id`);

setof `topoelement` **GetTopoGeomElements**(topogeometry `tg`);

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` 是 `topology.layer` 的 `layer_id` 的 `TopoGeometry` 的 `TopoGeometry ID`。

2.0.0 版本。

返回

返回

`GetTopoGeomElementArray`, `TopoElement`, `TopoGeom_addElement`, `TopoGeom_remElement`

9.11.3 ST_SRID

`ST_SRID` — Returns the spatial reference identifier for a `topogeometry`.

Synopsis

integer **ST_SRID**(topogeometry `tg`);

国国

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

国国

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
-- result
4326
```

国国

Section 4.5, [ST_SetSRID](#), [ST_Transform](#), [ST_SRID](#)

9.12 TopoGeometry 国国国

9.12.1 AsGML

AsGML — TopoGeometry 国 GML 国国国国国国国国国.

Synopsis

```
text AsGML(topogeometry tg);
text AsGML(topogeometry tg, text nsprefix_in);
text AsGML(topogeometry tg, regclass visitedTable);
text AsGML(topogeometry tg, regclass visitedTable, text nsprefix);
text AsGML(topogeometry tg, text nsprefix_in, integer precision, integer options);
text AsGML(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable);
text AsGML(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable,
text idprefix);
text AsGML(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable,
text idprefix, int gmlversion);
```



TopoGeometry ☐ GML ☐ GML3 ☐ . nsprefix_in ☐ gml ☐ . nsprefix ☐ (non-qualified) ☐ ☐ (15) ☐ (1) ☐ , ☐ ST_AsGML ☐ ☐.

[illegible]

```
CREATE TABLE visited (
    element_type integer, element_id integer,
    unique(element_type, element_id)
);
```

```
idprefix 00000000, 000000000000000000000000.
```

gm1ver 1.0.0, ST_AsGML 1.0.0. 3 2.0.0.



 CreateTopoGeom 

```
SELECT topology.AsGML(topo) As rdgml
FROM ri.roads
WHERE road name = 'Unknown';
```

```
-- rdgml --
```

```
<gml:TopoCurve>
  <gml:directedEdge>
    <gml:Edge gml:id="E1">
      <gml:directedNode orientation="-">
        <gml:Node gml:id="N1"/>
      </gml:directedNode>
    </gml:directedEdge>
  </gml:TopoCurve>
  <gml:curveProperty>
    <gml:Curve srsName="urn:ogc:def:crs:EPSG::3438">
      <gml:segments>
        <gml:LineStringSegment>
          <gml:posList srsDimension="2">
            384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
            384833 236884 384844 236882 384866 236881 384879 236883 384954
            236898 385087 236932 385117 236938
            385167 236938 385203 236941 385224 236946 385233 236950 385241
            236956 385254 236971
            385260 236979 385268 236999 385273 237018 385273 237037 385271
            237047 385267 237057 385225 237125
            385210 237144 385192 237161 385167 237192 385162 237202 385159
            237214 385159 237227 385162 237241
            385166 237256 385196 237324 385209 237345 385234 237375 385237
            237383 385238 237399 385236 237407
            385227 237419 385213 237430 385193 237439 385174 237451 385170
            237455 385169 237460 385171 237475
            385181 237503 385190 237521 385200 237533 385206 237538 385213
            237541 385221 237542 385235 237540 385242 237541
          </gml:posList>
        </gml:LineStringSegment>
      </gml:segments>
    </gml:Curve>
  </gml:curveProperty>
</gml:TopoCurve>
```

```

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>

```

创建拓扑几何。

```

SELECT topology.AsGML(topo,'') As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';

-- rdgml--
<TopoCurve>
  <directedEdge>
    <Edge id="E1">
      <directedNode orientation="-">
        <Node id="N1"/>
      </directedNode>
      <directedNode
></directedNode>
      <curveProperty>
        <Curve srsName="urn:ogc:def:crs:EPSG::3438">
          <segments>
            <LineStringSegment>
              <posList srsDimension="2">
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
384833 236884 384844 236882 384866 236881 384879 236883 384954 ←
236898 385087 236932 385117 236938
385167 236938 385203 236941 385224 236946 385233 236950 385241 ←
236956 385254 236971
385260 236979 385268 236999 385273 237018 385273 237037 385271 ←
237047 385267 237057 385225 237125
385210 237144 385192 237161 385167 237192 385162 237202 385159 ←
237214 385159 237227 385162 237241
385166 237256 385196 237324 385209 237345 385234 237375 385237 ←
237383 385238 237399 385236 237407
385227 237419 385213 237430 385193 237439 385174 237451 385170 ←
237455 385169 237460 385171 237475
385181 237503 385190 237521 385200 237533 385206 237538 385213 ←
237541 385221 237542 385235 237540 385242 237541
385249 237544 385260 237555 385270 237570 385289 237584 385292 ←
237589 385291 237596 385284 237630</posList>
            </LineStringSegment>
          </segments>
        </Curve>
      </curveProperty>
    </Edge>
  </directedEdge>
</TopoCurve>

```

创建

CreateTopoGeom, ST_CreateTopoGeo

9.12.2 AsTopoJSON

AsTopoJSON — TopoGeometry 到 TopoJSON 的转换器。

Synopsis

```
text AsTopoJSON(topogeometry tg, regclass edgeMapTable);
```

参数

TopoGeometry 到 TopoJSON 的转换器。 edgeMapTable 为 NULL 时，使用默认值 (arc) 的查找/存储 (lookup/storage) 格式。 compact 为 " " 时，使用紧凑格式。

返回的 TopoJSON 对象，包含 "serial" 和 "arc_id" 属性。 "edge_id" 属性是可选的。 "edge_id" 属性是可选的。



Note

TopoJSON 的 "edgeMapTable" 属性是 0-1 的浮点值。

TopoJSON 的 "type" 属性是 "Topology"。 "transform" 属性是 { "scale": [1,1], "translate": [0,0] }。 "objects" 属性是 TopoJSON 对象的数组。

2.1.0 版本支持 TopoJSON 的 "type" 属性。

版本: 2.2.1 版本 (puntal) 支持 TopoJSON 的 "type" 属性。

参数

ST_AsGeoJSON

参数

```
CREATE TEMP TABLE edgemap(arc_id serial, edge_id int unique);

-- header
SELECT '{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects": {'
```

```
-- objects
UNION ALL SELECT '' || feature_name || ': ' || AsTopoJSON(feature, 'edgemap')
FROM features.big_parcel WHERE feature_name = 'P3P4';

-- arcs
WITH edges AS (
  SELECT m.arc_id, e.geom FROM edgemap m, city_data.edge e
  WHERE e.edge_id = m.edge_id
), points AS (
  SELECT arc_id, (st_dumppoints(geom)).* FROM edges
), compare AS (
  SELECT p2.arc_id,
    CASE WHEN p1.path IS NULL THEN p2.geom
    ELSE ST_Translate(p2.geom, -ST_X(p1.geom), -ST_Y(p1.geom))
```

```

        END AS geom
FROM points p2 LEFT OUTER JOIN points p1
ON ( p1.arc_id = p2.arc_id AND p2.path[1] = p1.path[1]+1 )
ORDER BY arc_id, p2.path
), arcsdump 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 arcsdump
GROUP BY arc_id
ORDER BY arc_id
)
SELECT '}', "arcs": [' UNION ALL
SELECT array_to_string(array_agg(a), E',\n') from arcs

-- footer
UNION ALL SELECT '}]':::text as t;

-- Result:
{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects": {
"P3P4": { "type": "MultiPolygon", "arcs": [[[-1]],[[6,5,-5,-4,-3,1]]]}
}, "arcs": [
[[25,30],[6,0],[0,10],[-14,0],[0,-10],[8,0]],
[[35,6],[0,8]],
[[35,6],[12,0]],
[[47,6],[0,8]],
[[47,14],[0,8]],
[[35,22],[12,0]],
[[35,14],[0,8]]
]]
}

```

9.13 国际化

9.13.1 Equals

Equals — 检查 TopoGeometry 是否相等。

Synopsis

boolean **Equals**(topogeometry tg1, topogeometry tg2);

返回

检查 TopoGeometry 是否相等 (tg1, tg2, 0) 返回 0。



Note

检查 TopoGeometry 是否相等。检查 TopoGeometry 是否相等。

1.1.0 国际化。



This function supports 3d and will not drop the z-index.

函数

函数

[GetTopoGeomElements](#), [ST_Equals](#)

9.13.2 Intersects

Intersects — 检查 TopoGeometry 是否相交。

Synopsis

```
boolean Intersects(topogeometry tg1, topogeometry tg2);
```

函数

TopoGeometry 是否相交。



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 版本引入。



This function supports 3d and will not drop the z-index.

函数

函数

[ST_Intersects](#)

9.14 Importing and exporting Topologies

Once you have created topologies, and maybe associated topological layers, you might want to export them into a file-based format for backup or transfer into another database.

Using the standard dump/restore tools of PostgreSQL is problematic because topologies are composed by a set of tables (4 for primitives, an arbitrary number for layers) and records in metadata tables (topology.topology and topology.layer). Additionally, topology identifiers are not univoque across databases so that parameter of your topology will need to be changes upon restoring it.

In order to simplify export/restore of topologies a pair of executables are provided: `pgtopo_export` and `pgtopo_import`. Example usage:


```
pgtopo_export dev_db topo1 | pgtopo_import topo1 | psql staging_db
```

9.14.1 Using the Topology exporter

The `pgtopo_export` script takes the name of a database and a topology and outputs a dump file which can be used to import the topology (and associated layers) into a new database.

By default `pgtopo_export` writes the dump file to the standard output so that it can be piped to `pgtopo_import` or redirected to a file (refusing to write to terminal). You can optionally specify an output filename with the `-f` commandline switch.

By default `pgtopo_export` includes a dump of all layers defined against the given topology. This may be more data than you need, or may be non-working (in case your layer tables have complex dependencies) in which case you can request skipping the layers with the `--skip-layers` switch and deal with those separately.

Invoking `pgtopo_export` with the `--help` (or `-h` for short) switch will always print short usage string.

The dump file format is a compressed tar archive of a `pgtopo_export` directory containing at least a `pgtopo_dump_version` file with format version info. As of version 1 the directory contains tab-delimited CSV files with data of the topology primitive tables (node, edge_data, face, relation), the topology and layer records associated with it and (unless `--skip-layers` is given) a custom-format PostgreSQL dump of tables reported as being layers of the given topology.

9.14.2 Using the Topology importer

The `pgtopo_import` script takes a `pgtopo_export` format topology dump and a name to give to the topology to be created and outputs an SQL script reconstructing the topology and associated layers.

The generated SQL file will contain statements that create a topology with the given name, load primitive data in it, restores and registers all topology layers by properly linking all TopoGeometry values to their correct topology.

By default `pgtopo_import` reads the dump from the standard input so that it can be used in conjunction with `pgtopo_export` in a pipeline. You can optionally specify an input filename with the `-f` commandline switch.

By default `pgtopo_import` includes in the output SQL file the code to restore all layers found in the dump.

This may be unwanted or non-working in case your target database already have tables with the same name as the ones in the dump. In that case you can request skipping the layers with the `--skip-layers` switch and deal with those separately (or later).

SQL to only load and link layers to a named topology can be generated using the `--only-layers` switch. This can be useful to load layers AFTER resolving the naming conflicts or to link layers to a different topology (say a spatially-simplified version of the starting topology).

If the target topology already exists and you want it dropped upfront you can pass the `--drop-topology` switch (since PostGIS-3.6.0).

Chapter 10

栅格数据, 栅格索引

10.1 栅格数据

栅格数据, 栅格索引 raster2pgsql 是 PostGIS 的一个扩展, 用于加载栅格数据。

10.1.1 raster2pgsql 简介

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 安装与使用指南:

```
raster2pgsql -s 4326 -I -C -M *.tif -F -t 100x100 public.demelevation | psql -d gisdb
```

安装与使用指南: `aerial` 安装与使用指南, 安装 2 到 4 个
安装与使用指南, 安装与使用指南 (安装与使用指南 DB 安装) 安装, 安装与使用指南
安装与使用指南 -e 安装与使用指南 (安装与使用指南安装与使用指南安装与使用指南
安装与使用指南). 安装与使用指南 128x128 安装与使用指南安装与使用指南安装与使用指南
安装与使用指南. 安装与使用指南安装与使用指南安装与使用指南 -F 安装与使用指南“filename” 安装与使用指南
安装与使用指南.

```
raster2pgsql -I -C -e -Y -F -s 26986 -t 128x128 -l 2,4 bostonaerials2008/*.jpg aerals. ↵  
boston | psql -U postgres -d gisdb -h localhost -p 5432
```

--get a list of raster types supported:

```
raster2pgsql -G
```

-G 安装与使用指南:

Available GDAL raster formats:

```
Virtual Raster  
GeoTIFF  
National Imagery Transmission Format  
Raster Product Format TOC format  
ECRG TOC format  
Erdas Imagine Images (.img)  
CEOS SAR Image  
CEOS Image  
...  
Arc/Info Export E00 GRID  
ZMap Plus Grid  
NOAA NGS Geoid Height Grids
```

10.1.1.2 raster2pgsql options

-? 安装与使用指南. 安装与使用指南安装与使用指南安装与使用指南.

-G 安装与使用指南.

c|a|d|p -- 安装与使用指南:

- c 安装与使用指南 (安装与使用指南) 安装与使用指南. 安装与使用指南.
- a 安装与使用指南 (安装与使用指南) 安装与使用指南.
- d 安装与使用指南, 安装与使用指南安装与使用指南 (安装与使用指南) 安装与使用指南.
- p 安装与使用指南, 安装与使用指南.

安装与使用指南: 安装与使用指南安装与使用指南安装与使用指南

- C `raster_columns` 安装与使用指南安装与使用指南 SRID, 安装与使用指南安装与使用指南
安装与使用指南.
- x 安装与使用指南 (extent) 安装与使用指南安装与使用指南. -C 安装与使用指南安装与使用指南安装与使用指南.

-r **regular blocking** 使用正则阻塞 (regular blocking) 模式。-C 选项
指定正则阻塞的块大小。

选项: 指定要写入的表名。

-s <SRID> 指定 SRID 值。如果为 0，则使用 SRID 值。
指定要写入的表名。

-b BAND 指定要写入的波段 (1-255)。如果为 0，则使用所有波段。
指定要写入的表名。

-t TILE_SIZE 指定要写入的瓦片大小。TILE_SIZE 指定为 x 或 y
指定要写入的表名。

-P 指定要写入的瓦片大小 (padding) 指定要写入的表名。

-R, --register 指定要写入的表名 (DB 表) 指定要写入的表名。
指定要写入的表名。

-l OVERVIEW_FACTOR 指定要写入的表名。指定要写入的表名。
指定要写入的表名。

-N NODATA "NODATA" 指定要写入的表名 NODATA 指定要写入的表名。

选项: 指定要写入的表名。

-f COLUMN 指定要写入的表名。指定要写入的表名。

-F 指定要写入的表名。

-n COLUMN 指定要写入的表名。-F 指定要写入的表名。

-q PostgreSQL 指定要写入的表名。

-I GiST 指定要写入的表名。

-M (vacuum analyze) 指定要写入的表名。

-k Keeps empty tiles and skips NODATA value checks for each raster band. Note you save time
in checking, but could end up with far more junk rows in your database and those junk rows
are not marked as empty tiles.

-T tablespace 指定要写入的表名。-X 指定要写入的表名 (指定要写入的表名)

-X tablespace 指定要写入的表名。-I 指定要写入的表名。

-Y max_rows_per_copy=50 Use copy statements instead of insert statements. Optionally specify
max_rows_per_copy; default 50 when not specified.

-e 指定要写入的表名, 指定要写入的表名 (transaction) 指定要写入的表名。

-E ENDIAN 指定要写入的表名 (endianness) 指定要写入的表名。XDR 0, 指定
指定要写入的表名。NDR 1 指定要写入的表名。

-V version 指定要写入的表名。指定要写入的表名。0 指定要写入的表名。

10.1.2 PostGIS 安装与使用

指定要写入的表名。指定要写入的表名。

1. 指定要写入的表名:

```
CREATE TABLE myrasters(rid serial primary key, rast raster);
```

2. 创建空栅格。使用 `ST_MakeEmptyRaster` 函数，`ST_AddBand` 函数添加波段。
使用 `ST_AsRaster` 函数，`ST_Union`，`ST_MapAlgebra` (algebra) 函数。
使用 `ST_Transform` 函数。
3. 创建索引，使用 `ST_ConvexHull` 函数：

```
CREATE INDEX myrasters_rast_st_convexhull_idx ON myrasters USING gist( ST_ConvexHull( ←  
rast) );
```

(convex hull) 使用 `ST_ConvexHull` 函数。



Note

PostGIS 2.0 使用 `envelop` 函数。
使用 `ST_ConvexHull` 函数。

4. `AddRasterConstraints` 函数。

10.1.3 Using "out db" cloud rasters

The `raster2pgsql` tool uses GDAL to access raster data, and can take advantage of a key GDAL feature: the ability to read from rasters that are **stored remotely** in cloud "object stores" (e.g. AWS S3, Google Cloud Storage).

Efficient use of cloud stored rasters requires the use of a "cloud optimized" format. The most well-known and widely used is the "**cloud optimized GeoTIFF**" format. Using a non-cloud format, like a JPEG, or an un-tiled TIFF will result in very poor performance, as the system will have to download the entire raster each time it needs to access a subset.

First, load your raster into the cloud storage of your choice. Once it is loaded, you will have a URI to access it with, either an "http" URI, or sometimes a URI specific to the service. (e.g., "s3://bucket/object"). To access non-public buckets, you will need to supply GDAL config options to authenticate your connection. Note that this command is *reading* from the cloud raster and *writing* to the database.

```
AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxx \  
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \  
raster2pgsql \  
-s 990000 \  
-t 256x256 \  
-I \  
-R \  
/vsis3/your.bucket.com/your_file.tif \  
your_table \  
| psql your_db
```

Once the table is loaded, you need to give the database permission to read from remote rasters, by setting two permissions, `postgis.enable_outdb_rasters` and `postgis.gdal_enabled_drivers`.

```
SET postgis.enable_outdb_rasters = true;  
SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

To make the changes sticky, set them directly on your database. You will need to re-connect to experience the new settings.

```
ALTER DATABASE your_db SET postgis.enable_outdb_rasters = true;
ALTER DATABASE your_db SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

For non-public rasters, you may have to provide access keys to read from the cloud rasters. The same keys you used to write the `raster2pgsql` call can be set for use inside the database, with the `postgis.gdal_vsi_options` configuration. Note that multiple options can be set by space-separating the key=value pairs.

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
```

Once you have the data loaded and permissions set you can interact with the raster table like any other raster table, using the same functions. The database will handle all the mechanics of connecting to the cloud data when it needs to read pixel data.

10.2 使用云数据

PostGIS 支持使用云数据。使用云数据时，您需要提供访问密钥。这些密钥用于写入 `raster2pgsql` 调用，也可以设置在数据库内部使用，通过 `postgis.gdal_vsi_options` 配置。注意，多个选项可以通过空格分隔的 key=value 对来设置。

1. `raster_columns` 表用于存储云数据的元数据。
2. `raster_overviews` 表用于存储云数据的概图。使用 `-l` 选项可以生成概图。

10.2.1 使用云数据

`raster_columns` 表用于存储云数据的元数据。该表包含以下列：
`raster_id`：云数据的唯一标识符。
`table_name`：存储数据的表名。
`column_name`：存储数据的列名。
`srid`：空间参考标识符。
`scale_x`：X轴的比例尺。
`scale_y`：Y轴的比例尺。

使用 `-c` 选项可以生成概图。在 `raster_columns` 表中，`AddRasterConstraints` 列用于存储约束信息。

- `r_table_catalog`：存储数据的目录名。
- `r_table_schema`：存储数据的模式名。
- `r_table_name`：存储数据的表名。
- `r_raster_column`：存储数据的列名。PostGIS 使用 `raster_columns` 表来管理云数据。
- `srid`：空间参考标识符。Section 4.5 介绍了如何设置。
- `scale_x`：X轴的比例尺。使用 `scale_x` 和 `scale_y` 来设置。使用 `ST_ScaleX` 函数。

- `scale_y` 縦方向のスケール (倍率) を指定。デフォルトは `scale_y` が `scale_x` と同じ値になる。指定する場合は `ST_ScaleY` を参照。
- `blocksize_x` ブロックの横方向のサイズ (ピクセル単位) を指定。指定する場合は `ST_Width` を参照。
- `blocksize_y` ブロックの縦方向のサイズ (ピクセル単位) を指定。指定する場合は `ST_Height` を参照。
- `same_alignment` 出力のピクセルアライメントが元のピクセルアライメントと同じかどうかを指定。指定する場合は `ST_SameAlignment` を参照。
- `regular_blocking` 出力のピクセルが規則的に配置されるかどうかを指定。指定する場合は `ST_RegularBlocking` を参照。
- `num_bands` 出力のバンド数を指定。指定する場合は `ST_NumBands` を参照。
- `pixel_types` 出力のピクセルタイプを指定。指定する場合は `ST_BandPixelType` を参照。
- `nodata_values` ノodata 値 `nodata_value` (double precision) を指定 (数)。指定する場合は `ST_BandNoDataValue` を参照。
- `out_db` 出力のデータベースを指定。指定する場合は `ST_OutDB` を参照。
- `extent` 出力の範囲 (extent) を指定。指定する場合は `DropRasterConstraints` と `AddRasterConstraints` を参照。
- `spatial_index` 空間インデックスを指定。

10.2.2 概要

`raster_overviews` テーブルは、`raster_columns` テーブルの `raster_overviews` カラムにリンクされている。このリンクは、`raster_columns` テーブルの `raster_overviews` カラムに `-1` を指定することで、`AddOverviewConstraints` を実行することで作成される。

このリンクは、`raster_columns` テーブルの `raster_overviews` カラムに `-1` を指定することで、`AddOverviewConstraints` を実行することで作成される。



Note

`raster_overviews` テーブルは `raster_columns` テーブルの `raster_overviews` カラムにリンクされている。このリンクは、`raster_columns` テーブルの `raster_overviews` カラムに `-1` を指定することで、`AddOverviewConstraints` を実行することで作成される。

このリンクは、`raster_columns` テーブルの `raster_overviews` カラムに `-1` を指定することで、`AddOverviewConstraints` を実行することで作成される。

1. `raster_columns` テーブルの `raster_overviews` カラムに `-1` を指定する。
2. `raster_columns` テーブルの `raster_overviews` カラムに `-1` を指定する。指定する場合は `DropRasterConstraints` と `AddRasterConstraints` を参照。

`raster_overviews` テーブルは、`raster_columns` テーブルの `raster_overviews` カラムにリンクされている。

- `o_table_catalog` 表所在数据库名称。
- `o_table_schema` 表所在模式名称。
- `o_table_name` 表名称。
- `o_raster_column` 栅格列名称。
- `r_table_catalog` 表所在数据库名称。
- `r_table_schema` 表所在模式名称。
- `r_table_name` 表名称。
- `r_raster_column` 栅格列名称。
- `overview_factor` 概览因子。该因子决定了在生成概览图时，原始栅格的分辨率被缩小了多少倍。例如，如果原始栅格的分辨率为 1 米，那么因子为 2 的概览图的分辨率就是 2 米。因子为 4 的概览图的分辨率就是 4 米。因子为 125 的概览图的分辨率就是 125 米。因子为 5000 的概览图的分辨率就是 5000 米。因子为 1600 的概览图的分辨率就是 1600 米。因子为 400 的概览图的分辨率就是 400 米。因子为 200 的概览图的分辨率就是 200 米。因子为 100 的概览图的分辨率就是 100 米。因子为 50 的概览图的分辨率就是 50 米。因子为 25 的概览图的分辨率就是 25 米。因子为 12.5 的概览图的分辨率就是 12.5 米。因子为 6.25 的概览图的分辨率就是 6.25 米。因子为 3.125 的概览图的分辨率就是 3.125 米。因子为 1.5625 的概览图的分辨率就是 1.5625 米。因子为 0.78125 的概览图的分辨率就是 0.78125 米。因子为 0.390625 的概览图的分辨率就是 0.390625 米。因子为 0.1953125 的概览图的分辨率就是 0.1953125 米。因子为 0.09765625 的概览图的分辨率就是 0.09765625 米。因子为 0.048828125 的概览图的分辨率就是 0.048828125 米。因子为 0.0244140625 的概览图的分辨率就是 0.0244140625 米。因子为 0.01220703125 的概览图的分辨率就是 0.01220703125 米。因子为 0.006103515625 的概览图的分辨率就是 0.006103515625 米。因子为 0.0030517578125 的概览图的分辨率就是 0.0030517578125 米。因子为 0.00152587890625 的概览图的分辨率就是 0.00152587890625 米。因子为 0.000762939453125 的概览图的分辨率就是 0.000762939453125 米。因子为 0.0003814697265625 的概览图的分辨率就是 0.0003814697265625 米。因子为 0.00019073486328125 的概览图的分辨率就是 0.00019073486328125 米。因子为 9.5367431640625e-05 的概览图的分辨率就是 9.5367431640625e-05 米。因子为 4.76837158203125e-05 的概览图的分辨率就是 4.76837158203125e-05 米。因子为 2.384185791015625e-05 的概览图的分辨率就是 2.384185791015625e-05 米。因子为 1.1920928955078125e-05 的概览图的分辨率就是 1.1920928955078125e-05 米。因子为 5.9604644775390625e-06 的概览图的分辨率就是 5.9604644775390625e-06 米。因子为 2.98023223876953125e-06 的概览图的分辨率就是 2.98023223876953125e-06 米。因子为 1.490116119384765625e-06 的概览图的分辨率就是 1.490116119384765625e-06 米。因子为 7.450580596923828125e-07 的概览图的分辨率就是 7.450580596923828125e-07 米。因子为 3.7252902984619140625e-07 的概览图的分辨率就是 3.7252902984619140625e-07 米。因子为 1.86264514923095703125e-07 的概览图的分辨率就是 1.86264514923095703125e-07 米。因子为 9.31322574615478515625e-08 的概览图的分辨率就是 9.31322574615478515625e-08 米。因子为 4.656612873077392578125e-08 的概览图的分辨率就是 4.656612873077392578125e-08 米。因子为 2.3283064365386962890625e-08 的概览图的分辨率就是 2.3283064365386962890625e-08 米。因子为 1.16415321826934814453125e-08 的概览图的分辨率就是 1.16415321826934814453125e-08 米。因子为 5.82076609134674072265625e-09 的概览图的分辨率就是 5.82076609134674072265625e-09 米。因子为 2.910383045673370361328125e-09 的概览图的分辨率就是 2.910383045673370361328125e-09 米。因子为 1.4551915228366851806640625e-09 的概览图的分辨率就是 1.4551915228366851806640625e-09 米。因子为 7.2759576141834259033203125e-10 的概览图的分辨率就是 7.2759576141834259033203125e-10 米。因子为 3.63797880709171295166015625e-10 的概览图的分辨率就是 3.63797880709171295166015625e-10 米。因子为 1.818989403545856475830078125e-10 的概览图的分辨率就是 1.818989403545856475830078125e-10 米。因子为 9.094947017729282379150390625e-11 的概览图的分辨率就是 9.094947017729282379150390625e-11 米。因子为 4.5474735088646411895751953125e-11 的概览图的分辨率就是 4.5474735088646411895751953125e-11 米。因子为 2.27373675443232059478759765625e-11 的概览图的分辨率就是 2.27373675443232059478759765625e-11 米。因子为 1.136868377216160297393798828125e-11 的概览图的分辨率就是 1.136868377216160297393798828125e-11 米。因子为 5.684341886080801486968994140625e-12 的概览图的分辨率就是 5.684341886080801486968994140625e-12 米。因子为 2.8421709430404007434844970703125e-12 的概览图的分辨率就是 2.8421709430404007434844970703125e-12 米。因子为 1.42108547152020037174224853515625e-12 的概览图的分辨率就是 1.42108547152020037174224853515625e-12 米。因子为 7.10542735760100185871124267578125e-13 的概览图的分辨率就是 7.10542735760100185871124267578125e-13 米。因子为 3.552713678800500929355621337890625e-13 的概览图的分辨率就是 3.552713678800500929355621337890625e-13 米。因子为 1.7763568394002504646778106689453125e-13 的概览图的分辨率就是 1.7763568394002504646778106689453125e-13 米。因子为 8.8817841970012523233890533447265625e-14 的概览图的分辨率就是 8.8817841970012523233890533447265625e-14 米。因子为 4.44089209850062616169452667236328125e-14 的概览图的分辨率就是 4.44089209850062616169452667236328125e-14 米。因子为 2.220446049250313080847263336181640625e-14 的概览图的分辨率就是 2.220446049250313080847263336181640625e-14 米。因子为 1.1102230246251565404236316680908203125e-14 的概览图的分辨率就是 1.1102230246251565404236316680908203125e-14 米。因子为 5.5511151231257827021181583340541015625e-15 的概览图的分辨率就是 5.5511151231257827021181583340541015625e-15 米。因子为 2.77555756156289135105907916702705078125e-15 的概览图的分辨率就是 2.77555756156289135105907916702705078125e-15 米。因子为 1.387778780781445675529539583513525390625e-15 的概览图的分辨率就是 1.387778780781445675529539583513525390625e-15 米。因子为 6.938893903907228377647697917567626953125e-16 的概览图的分辨率就是 6.938893903907228377647697917567626953125e-16 米。因子为 3.4694469519536141888238489587838134765625e-16 的概览图的分辨率就是 3.4694469519536141888238489587838134765625e-16 米。因子为 1.73472347597680709441192447939190673828125e-16 的概览图的分辨率就是 1.73472347597680709441192447939190673828125e-16 米。因子为 8.67361737988403547205962239695953369140625e-17 的概览图的分辨率就是 8.67361737988403547205962239695953369140625e-17 米。因子为 4.336808689942017736029811198479766845703125e-17 的概览图的分辨率就是 4.336808689942017736029811198479766845703125e-17 米。因子为 2.1684043449710088680149055992398834228515625e-17 的概览图的分辨率就是 2.1684043449710088680149055992398834228515625e-17 米。因子为 1.08420217248550443400745279961994171142578125e-17 的概览图的分辨率就是 1.08420217248550443400745279961994171142578125e-17 米。因子为 5.42101086242752217003726399809970855712890625e-18 的概览图的分辨率就是 5.42101086242752217003726399809970855712890625e-18 米。因子为 2.710505431213761085018631999049854278564453125e-18 的概览图的分辨率就是 2.710505431213761085018631999049854278564453125e-18 米。因子为 1.3552527156068805425093159995249271392822265625e-18 的概览图的分辨率就是 1.3552527156068805425093159995249271392822265625e-18 米。因子为 6.7762635780344027125465799976246356964111328125e-19 的概览图的分辨率就是 6.7762635780344027125465799976246356964111328125e-19 米。因子为 3.38813178901720135627328999881231784820556640625e-19 的概览图的分辨率就是 3.38813178901720135627328999881231784820556640625e-19 米。因子为 1.694065894508600678136644999406158924102783203125e-19 的概览图的分辨率就是 1.694065894508600678136644999406158924102783203125e-19 米。因子为 8.470329472543003390683224997030794620513916015625e-20 的概览图的分辨率就是 8.470329472543003390683224997030794620513916015625e-20 米。因子为 4.2351647362715016953416124985153973102569580078125e-20 的概览图的分辨率就是 4.2351647362715016953416124985153973102569580078125e-20 米。因子为 2.11758236813575084767080624925769865512847900390625e-20 的概览图的分辨率就是 2.11758236813575084767080624925769865512847900390625e-20 米。因子为 1.058791184067875423835403124628849327564239501953125e-20 的概览图的分辨率就是 1.058791184067875423835403124628849327564239501953125e-20 米。因子为 5.293955920339377119177015623144246637821197509765625e-21 的概览图的分辨率就是 5.293955920339377119177015623144246637821197509765625e-21 米。因子为 2.6469779601696885595885078115721233189105987548828125e-21 的概览图的分辨率就是 2.6469779601696885595885078115721233189105987548828125e-21 米。因子为 1.32348898008484427979425390578606165945529937744140625e-21 的概览图的分辨率就是 1.32348898008484427979425390578606165945529937744140625e-21 米。因子为 6.61744490042422139897126952893030829727649688720703125e-22 的概览图的分辨率就是 6.61744490042422139897126952893030829727649688720703125e-22 米。因子为 3.308722450212110699485634764465154148638248443603515625e-22 的概览图的分辨率就是 3.308722450212110699485634764465154148638248443603515625e-22 米。因子为 1.6543612251060553497428173822325770743191242218017578125e-22 的概览图的分辨率就是 1.6543612251060553497428173822325770743191242218017578125e-22 米。因子为 8.2718061255302767487140869111628853715956211090087890625e-23 的概览图的分辨率就是 8.2718061255302767487140869111628853715956211090087890625e-23 米。因子为 4.13590306276513837435704345558144268579781055450439453125e-23 的概览图的分辨率就是 4.13590306276513837435704345558144268579781055450439453125e-23 米。因子为 2.067951531382569187178521727790721342898905277252197265625e-23 的概览图的分辨率就是 2.067951531382569187178521727790721342898905277252197265625e-23 米。因子为 1.0339757656912845935892608638953606714494526386260986328125e-23 的概览图的分辨率就是 1.0339757656912845935892608638953606714494526386260986328125e-23 米。因子为 5.1698788284564229679463043169768033572472631931304931640625e-24 的概览图的分辨率就是 5.1698788284564229679463043169768033572472631931304931640625e-24 米。因子为 2.58493941422821148397315215848840167862363159656524658203125e-24 的概览图的分辨率就是 2.58493941422821148397315215848840167862363159656524658203125e-24 米。因子为 1.292469707114105741986576079244200839311815798282623291015625e-24 的概览图的分辨率就是 1.292469707114105741986576079244200839311815798282623291015625e-24 米。因子为 6.462348535570528709932880396221004196559078991413116455078125e-25 的概览图的分辨率就是 6.462348535570528709932880396221004196559078991413116455078125e-25 米。因子为 3.2311742677852643549664401981105020982795394957065582275390625e-25 的概览图的分辨率就是 3.2311742677852643549664401981105020982795394957065582275390625e-25 米。因子为 1.61558713389263217748322009905525104913976974785327911376953125e-25 的概览图的分辨率就是 1.61558713389263217748322009905525104913976974785327911376953125e-25 米。因子为 8.07793566946316088741610049527625524569884873926639556884765625e-26 的概览图的分辨率就是 8.07793566946316088741610049527625524569884873926639556884765625e-26 米。因子为 4.038967834731580443708050247638127622849424369633197784423828125e-26 的概览图的分辨率就是 4.038967834731580443708050247638127622849424369633197784423828125e-26 米。因子为 2.0194839173657902218540251238190638114247121848165988922119140625e-26 的概览图的分辨率就是 2.0194839173657902218540251238190638114247121848165988922119140625e-26 米。因子为 1.00974195868289511092701256190953190571235609240829944610595703125e-26 的概览图的分辨率就是 1.00974195868289511092701256190953190571235609240829944610595703125e-26 米。因子为 5.0487097934144755546350628095476595285617804620414972305297875e-27 的概览图的分辨率就是 5.0487097934144755546350628095476595285617804620414972305297875e-27 米。因子为 2.52435489670723777731753140477382976428089023102074861526489375e-27 的概览图的分辨率就是 2.52435489670723777731753140477382976428089023102074861526489375e-27 米。因子为 1.262177448353618888658765702386914882140445115510374307632446875e-27 的概览图的分辨率就是 1.262177448353618888658765702386914882140445115510374307632446875e-27 米。因子为 6.310887241768094443293828511934574410702225577551871538162234375e-28 的概览图的分辨率就是 6.310887241768094443293828511934574410702225577551871538162234375e-28 米。因子为 3.1554436208840472216469142559672872053511127887759357690811171875e-28 的概览图的分辨率就是 3.1554436208840472216469142559672872053511127887759357690811171875e-28 米。因子为 1.57772181044202361082345712798364360267555639438796788454055859375e-28 的概览图的分辨率就是 1.57772181044202361082345712798364360267555639438796788454055859375e-28 米。因子为 7.88860905221011805411728563991821801337778197193983942270279296875e-29 的概览图的分辨率就是 7.88860905221011805411728563991821801337778197193983942270279296875e-29 米。因子为 3.944304526105059027058642819959109006688890985969919711351396484375e-29 的概览图的分辨率就是 3.944304526105059027058642819959109006688890985969919711351396484375e-29 米。因子为 1.9721522630525295135293214099795545033444454929849598556756982421875e-29 的概览图的分辨率就是 1.9721522630525295135293214099795545033444454929849598556756982421875e-29 米。因子为 9.8607613152626475676466070498977725167222274649247992783784912109375e-30 的概览图的分辨率就是 9.8607613152626475676466070498977725167222274649247992783784912109375e-30 米。因子为 4.93038065763132378382330352494888625836111373246239963918924560546875e-30 的概览图的分辨率就是 4.93038065763132378382330352494888625836111373246239963918924560546875e-30 米。因子为 2.465190328815661891911651762474443129180556866231199819594622802734375e-30 的概览图的分辨率就是 2.465190328815661891911651762474443129180556866231199819594622802734375e-30 米。因子为 1.2325951644078309459558258812372215645902784331155999097973114013671875e-30 的概览图的分辨率就是 1.2325951644078309459558258812372215645902784331155999097973114013671875e-30 米。因子为 6.1629758220391547297791279061861078229513921655779995489865570068359375e-31 的概览图的分辨率就是 6.1629758220391547297791279061861078229513921655779995489865570068359375e-31 米。因子为 3.08148791101957736488956395309305391147569608278899977449327850341796875e-31 的概览图的分辨率就是 3.08148791101957736488956395309305391147569608278899977449327850341796875e-31 米。因子为 1.540743955509788682444781976546526955737848041394499887246639251708984375e-31 的概览图的分辨率就是 1.540743955509788682444781976546526955737848041394499887246639251708984375e-31 米。因子为 7.703719777548943412223909882732634778689240206972499436233196258544921875e-32 的概览图的分辨率就是 7.703719777548943412223909882732634778689240206972499436233196258544921875e-32 米。因子为 3.8518598887744717061119549413663173893446201034862497181165981292724609375e-32 的概览图的分辨率就是 3.8518598887744717061119549413663173893446201034862497181165981292724609375e-32 米。因子为 1.92592994438723585305597747068315869467231005174312485905829906463623046875e-32 的概览图的分辨率就是 1.92592994438723585305597747068315869467231005174312485905829906463623046875e-32 米。因子为 9.62964972193617926527988735341579347336155025871562429529149532318115234375e-32 的概览图的分辨率就是 9.62964972193617926527988735341579347336155025871562429529149532318115234375e-32 米。因子为 4.814824860968089632639943676707896736680775129357812147645747661590576171875e-32 的概览图的分辨率就是 4.814824860968089632639943676707896736680775129357812147645747661590576171875e-32 米。因子为 2.4074124304840448163199718383539483683403875646789060738228738307952880859375e-32 的概览图的分辨率就是 2.40741243048404481631997183835394836834038756467890607382287


```

        $input_srid = intval($_REQUEST['srid']);
    }
    else { $input_srid = 26986; }
    /** The set bytea_output may be needed for PostgreSQL 9.0+, but not for 8.4 */
    $sql = "set bytea_output='escape';
    SELECT ST_AsPNG(ST_Transform(
        ST_AddBand(ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast,3)]),
        ,3)))
    FROM aerials.boston
    WHERE
        ST_Intersects(rast, ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, -71.1210, 42.218,4326),26986) );";
    $result = pg_query($sql);
    $row = pg_fetch_row($result);
    pg_free_result($result);
    if ($row === false) return;
    echo pg_unescape_bytea($row[0]);
    ?>

```

10.3.2 使用 ST_AsPNG 在 ASP.NET C# 中显示栅格

在 `nppgsql` PostgreSQL .NET 驱动程序中，`ST_AsGDALRaster` 函数可以接受 1, 2, 3 个 PHP 请求流 (request stream) 参数。PHP 代码中 `"img src"` HTML 属性可以指向该 URL。

在 `nppgsql` PostgreSQL .NET 驱动程序中，<http://nppgsql.projects.postgresql.org/> 提供了详细的文档。ASP.NET bin 目录下提供了示例代码。

`ST_Union` (combine) 函数将 WGS84 坐标系中的栅格合并，`ST_Transform` 函数将栅格转换为指定坐标系，`ST_AsPNG` 函数将栅格转换为 PNG 格式。

C# 代码示例请参考 Section 10.3.1 中的代码。

http://mywebserver/test_raster.php?srid=2249

在 `web.config` 文件中配置数据库连接字符串。

```

-- web.config connection string section --
<connectionStrings>
  <add name="DSN"
    connectionString="server=localhost;database=mydb;Port=5432;User Id=myuser;password=
    mypwd"/>
</connectionStrings>

```

```

// Code for TestRaster.ashx
<%@ WebHandler Language="C#" Class="TestRaster" %>
using System;
using System.Data;
using System.Web;
using Npgsql;

public class TestRaster : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        context.Response.ContentType = "image/png";
    }
}

```

```

        context.Response.BinaryWrite(GetResults(context));
    }

    public bool IsReusable {
        get { return false; }
    }

    public byte[] GetResults(HttpContext context)
    {
        byte[] result = null;
        NpgsqlCommand command;
        string sql = null;
        int input_srid = 26986;
    try {
        using (NpgsqlConnection conn = new NpgsqlConnection(System.↵
            Configuration.ConfigurationManager.ConnectionStrings["DSN"].↵
            ConnectionString)) {
            conn.Open();

            if (context.Request["srid"] != null)
            {
                input_srid = Convert.ToInt32(context.Request["srid"]);
            }
            sql = @"SELECT ST_AsPNG(
                    ST_Transform(
                        ST_AddBand(
                            ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast,3)]
                                ,:input_srid) ) As new_rast
                    FROM aerials.boston
                    WHERE
                        ST_Intersects(rast,
                            ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, ↵
                                -71.1210, 42.218,4326),26986) )");
            command = new NpgsqlCommand(sql, conn);
            command.Parameters.Add(new NpgsqlParameter("input_srid", input_srid));

            result = (byte[]) command.ExecuteScalar();
            conn.Close();
        }
    }
    catch (Exception ex)
    {
        result = null;
        context.Response.Write(ex.Message.Trim());
    }
    return result;
}

```

10.3.3 使用 Npgsql 在 Java 中访问 PostGIS

本节将介绍如何在 Java 中使用 Npgsql 访问 PostGIS。

<http://jdbc.postgresql.org/download.html> 提供了 PostgreSQL JDBC 驱动程序的下载链接。

安装驱动程序后，可以在 Java 代码中使用：

```
set env CLASSPATH ../../postgresql-9.0-801.jdbc4.jar
javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class
```

编译并打包后的文件如下：

```
java -jar SaveQueryImage.jar "SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10, ' ↵
quad_segs=2'),150, 150, '8BUI',100));" "test.png"
```

```
-- Manifest.txt --
Class-Path: postgresql-9.0-801.jdbc4.jar
Main-Class: SaveQueryImage
```

```
// Code for SaveQueryImage.java
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.*;

public class SaveQueryImage {
    public static void main(String[] argv) {
        System.out.println("Checking if Driver is registered with DriverManager.");

        try {
            //java.sql.DriverManager.registerDriver (new org.postgresql.Driver());
            Class.forName("org.postgresql.Driver");
        }
        catch (ClassNotFoundException cnfe) {
            System.out.println("Couldn't find the driver!");
            cnfe.printStackTrace();
            System.exit(1);
        }

        Connection conn = null;

        try {
            conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/mydb","myuser ↵
            ", "mypwd");
            conn.setAutoCommit(false);

            PreparedStatement sGetImg = conn.prepareStatement(argv[0]);

            ResultSet rs = sGetImg.executeQuery();

            FileOutputStream fout;
            try
            {
                rs.next();
                /** Output to file name requested by user */
                fout = new FileOutputStream(new File(argv[1]) );
                fout.write(rs.getBytes(1));
                fout.close();
            }
            catch(Exception e)
            {
                System.out.println("Can't create file");
                e.printStackTrace();
            }

            rs.close();
```

```

        sGetImg.close();
        conn.close();
    }
    catch (SQLException se) {
        System.out.println("Couldn't connect: print out a stack trace and exit.");
        se.printStackTrace();
        System.exit(1);
    }
}
}

```

10.3.4 PLPython 调用 SQL 函数

PLPython 是 PostgreSQL 的一个扩展。PLPython 函数使用 Python 语言编写。PLPython 函数使用 PLPython3u 扩展。

```

CREATE OR REPLACE FUNCTION write_file (param_bytes bytea, param_filepath text)
RETURNS text
AS $$
f = open(param_filepath, 'wb+')
f.write(param_bytes)
return param_filepath
$$ LANGUAGE plpythonu;

```

```

--write out 5 images to the PostgreSQL server in varying sizes
-- note the postgresql daemon account needs to have write access to folder
-- this echos back the file names created;
SELECT write_file(ST_AsPNG(
    ST_AsRaster(ST_Buffer(ST_Point(1,5),j*5, 'quad_segs=2'),150*j, 150*j, '8BUI',100)),
    'C:/temp/slices'|| j || '.png')
FROM generate_series(1,5) As j;

write_file
-----
C:/temp/slices1.png
C:/temp/slices2.png
C:/temp/slices3.png
C:/temp/slices4.png
C:/temp/slices5.png

```

10.3.5 PSQL 调用 PLPython 函数

PSQL 是 PostgreSQL 的交互式 SQL 客户端。PostgreSQL 的 PSQL 客户端可以调用 PLPython 函数。PSQL 客户端可以调用 PLPython 函数。

PSQL 客户端可以调用 PLPython 函数。

```

SELECT oid, lowrite(lo_open(oid, 131072), png) As num_bytes
FROM
( VALUES (lo_create(0),
    ST_AsPNG( (SELECT rast FROM aerials.boston WHERE rid=1) )
) ) As v(oid,png);
-- you'll get an output something like --
oid | num_bytes
-----+-----
2630819 | 74860

```

```
-- next note the oid and do this replacing the c:/test.png to file path location
-- on your local computer
\lo_export 2630819 'C:/temp/aerial_samp.png'

-- this deletes the file from large object storage on db
SELECT lo_unlink(2630819);
```


11.1 几何函数

11.1.1 geomval

geomval — (geometry) geom 的 (geometry) val, 返回 val 的几何类型。

例

```
geomval (SELECT geom, .geom FROM ST_DumpAsPolygon(
  (SELECT geom FROM table) val)) AS ST_DumpAsPolygon
  (SELECT geom FROM table) val)) AS ST_DumpAsPolygon
```

例

Section 13.6

11.1.2 addbandarg

addbandarg — 返回 ST_AddBand 的几何类型。

例

返回 ST_AddBand 的几何类型。

index integer 返回 1-1000。 NULL 返回, 返回 1000。

pixeltype text **ST_BandPixelType** 返回像素类型。

initialvalue double precision 返回初始值。

nodataval double precision 返回 NODATA 值。 NULL 返回, 返回 NODATA 值。

例

ST_AddBand

11.1.3 rastbandarg

rastbandarg — 返回 rastbandarg 的几何类型。

例

返回 rastbandarg 的几何类型。

rast raster 返回 rast 的几何类型。

nband integer 返回 1-1000。

栅格

ST_MapAlgebra (callback function version)

11.1.4 raster

raster — 栅格数据类型。

栅格

raster is a spatial data type used to represent raster data such as those imported from JPEGs, TIFFs, PNGs, digital elevation models. Each raster has 1 or more bands each having a set of pixel values. Rasters can be georeferenced.



Note
GDAL 与 PostGIS 集成。栅格数据可以存储在 PostGIS 中，使用 `ST_ConvexHull` 函数。栅格数据可以存储在 PostGIS 中，使用 `ST_ConvexHull` 函数。

栅格

栅格数据类型用于表示栅格数据，如从 JPEGs、TIFFs、PNGs、数字高程模型导入的数据。每个栅格有 1 个或多个波段，每个波段都有一组像素值。栅格可以地理参考。

栅格	栅格
栅格	栅格

栅格

Chapter 11

11.1.5 reclassarg

reclassarg — A composite type used as input into the ST_Reclass function defining the behavior of reclassification.

栅格

A composite type used as input into the `ST_Reclass` function defining the behavior of reclassification.

nband integer 栅格数据类型。

reclassexpr text 栅格数据类型。range:map_range 栅格数据类型。': ' 栅格数据类型。'(' > ' , ') ' 栅格数据类型，'] ' < ' 栅格数据类型， '[' > ' 栅格数据类型。

- 1. [a-b] = a <= x <= b
- 2. (a-b) = a < x <= b

3. $[a-b] = a \leq x < b$

4. $(a-b) = a < x < b$

'(' 返回 (a-b) 的 a-b 的符号。

pixeltype text **ST_BandPixelType** 返回像素类型。

nodataval double precision NODATA 返回 NODATA 值。返回 NODATA 值，返回 NODATA 值。

例: 返回 2 到 255 的 NODATA 值 8BUI 的符号。

```
SELECT ROW(2, '0-100:1-10, 101-500:11-150,501 - 10000: 151-254', '8BUI', 255)::reclassarg;
```

例: 返回 1 的 NODATA 值 1BB 的符号。

```
SELECT ROW(1, '0-100]:0, (100-255:1', '1BB', NULL)::reclassarg;
```

例

ST_Reclass

11.1.6 summarystats

summarystats — ST_SummaryStats 和 ST_SummaryStatsAgg 返回统计摘要。

例

ST_SummaryStats 和 **ST_SummaryStatsAgg** 返回统计摘要。

count integer 返回计数。

sum double precision 返回总和。

mean double precision 返回平均值。

stddev double precision 返回标准差。

min double precision 返回最小值。

max double precision 返回最大值。

例

ST_SummaryStats, **ST_SummaryStatsAgg**

11.1.7 unionarg

unionarg — 返回 UNION 的 ST_Union 返回结果。

##

ST_Union 函数用于将多个几何对象合并为一个单一的几何对象。

nband integer 指定输出栅格的波段数，默认为 1。

uniontype text 指定输出栅格的类型，默认为 **ST_Union**。

##

ST_Union

11.2 栅格函数

11.2.1 AddRasterConstraints

AddRasterConstraints — Adds raster constraints to a loaded raster table for a specific column that constrains spatial ref, scaling, blocksize, alignment, bands, band type and a flag to denote if raster column is regularly blocked. The table must be loaded with data for the constraints to be inferred. Returns true if the constraint setting was accomplished and issues a notice otherwise.

Synopsis

```
boolean AddRasterConstraints(name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true,
boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true,
boolean regular_blocking=false, boolean num_bands=true, boolean pixel_types=true, boolean no-
data_values=true, boolean out_db=true, boolean extent=true );
boolean AddRasterConstraints(name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean AddRasterConstraints(name rastschema, name rasttable, name rastcolumn, text[] VARI-
ADIC constraints);
boolean AddRasterConstraints(name rastschema, name rasttable, name rastcolumn, boolean srid=true,
boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true,
boolean same_alignment=true, boolean regular_blocking=false, boolean num_bands=true, boolean
pixel_types=true, boolean nodata_values=true, boolean out_db=true, boolean extent=true );
```

##

raster_columns 表用于存储栅格表的信息，包括表名、列名、SRID、块大小、对齐方式、波段数、像素类型、是否包含空值、是否包含延伸信息等。

raster2pgsql 函数用于将栅格数据导入到数据库中。

有关该函数的详细信息，请参考 Section 10.2.1。

- **blocksize** X Y 指定栅格的块大小。
- **blocksize_x** X 指定栅格的块大小（X 方向）。
- **blocksize_y** Y 指定栅格的块大小（Y 方向）。
- **extent** 指定栅格的延伸范围。
- **num_bands** 指定栅格的波段数。

- `pixel_types` 指定栅格像素的数据类型。指定 N 指定栅格像素的数据类型。
- `regular_blocking` 指定栅格是否规则 (指定栅格是否规则) 指定栅格是否规则。
- `same_alignment` ensures they all have same alignment meaning any two tiles you compare will return true for. Refer to [ST_SameAlignment](#).
- `srid` 指定栅格的 SRID 指定栅格的 SRID。
- `--` 指定栅格的 SRID 指定栅格的 SRID。

**Note**

指定栅格的 SRID 指定栅格的 SRID。指定栅格的 SRID 指定栅格的 SRID。

**Note**

指定栅格的 SRID 指定栅格的 SRID。指定栅格的 SRID 指定栅格的 SRID。

2.0.0 安装与配置

创建表 myrasters:

```
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}
```

创建表 myrasters2:

```
CREATE TABLE public.myrasters2(rid SERIAL primary key, rast raster);
INSERT INTO myrasters2(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, '8BSI'::
text, -129, NULL);
```

```
SELECT AddRasterConstraints('public'::name, 'myrasters2'::name, 'rast'::name, ' ←
    regular_blocking', 'blocksize');
-- get notice--
NOTICE: Adding regular blocking constraint
NOTICE: Adding blocksize-X constraint
NOTICE: Adding blocksize-Y constraint
```

☐☐

Section [10.2.1, ST_AddBand, ST_MakeEmptyRaster, DropRasterConstraints, ST_BandPixelType, ST_SRID](#)

11.2.2 DropRasterConstraints

DropRasterConstraints — 删除 PostGIS 栅格表上的约束。该函数删除指定的栅格表上的所有约束。

Synopsis

boolean **DropRasterConstraints**(name rasttable, name rastcolumn, boolean srid, boolean scale_x, boolean scale_y, boolean blocksize_x, boolean blocksize_y, boolean same_alignment, boolean regular_blocking, boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true, boolean out_db=true, boolean extent=true);

boolean **DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=false, boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true, boolean out_db=true, boolean extent=true);

boolean **DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, text[] constraints);

☐☐

AddRasterConstraints 函数，删除 PostGIS 栅格表上的约束。该函数删除指定的栅格表上的所有约束。

删除栅格表上的约束。

```
DROP TABLE mytable
```

删除栅格表上的约束，SQL 语句如下。

```
ALTER TABLE mytable DROP COLUMN rast
```

删除栅格表上的约束 raster_columns 函数。删除栅格表上的约束 raster_columns 函数。删除栅格表上的约束 raster_columns 函数。

2.0.0 版本。


```

    ST_MakeEmptyRaster(500, 500, 0, 0, 4),
    1, '8BSI'::text, -129, NULL
) r2;

SELECT AddOverviewConstraints('res2', 'r2', 'res1', 'r1', 2);

-- verify if registered correctly in the raster_overviews view --
SELECT o_table_name ot, o_raster_column oc,
       r_table_name rt, r_raster_column rc,
       overview_factor f
FROM raster_overviews WHERE o_table_name = 'res2';
  ot | oc | rt | rc | f
-----+-----+-----+-----+---
res2 | r2 | res1 | r1 | 2
(1 row)

```

☐☐

Section [10.2.2, DropOverviewConstraints, ST_CreateOverview, AddRasterConstraints](#)

11.2.4 DropOverviewConstraints

DropOverviewConstraints — 删除 (overview) 表中的概览数据。

Synopsis

boolean **DropOverviewConstraints**(name ovschema, name ovtable, name ovcolumn);
 boolean **DropOverviewConstraints**(name ovtable, name ovcolumn);

☐☐

raster_overviews 表中的概览数据。

ovschema 数据库名, search_path 搜索路径。

2.0.0 版本。

☐☐

Section [10.2.2, AddOverviewConstraints, DropRasterConstraints](#)

11.2.5 PostGIS_GDAL_Version

PostGIS_GDAL_Version — 返回 PostGIS 使用的 GDAL 版本。

Synopsis

text **PostGIS_GDAL_Version**();

安装

PostGIS 3.6.1 依赖于 GDAL 3.6.1 或更高版本。GDAL 3.6.1 或更高版本在大多数 Linux 发行版中默认安装。如果未安装，请安装 GDAL 3.6.1 或更高版本。

安装

```
SELECT PostGIS_GDAL_Version();
       postgis_gdal_version
-----
GDAL 1.11dev, released 2013/04/13
```

安装

`postgis.gdal_datapath`

11.2.6 PostGIS_Raster_Lib_Build_Date

PostGIS_Raster_Lib_Build_Date — 返回 PostGIS 的 Raster 库的构建日期。

Synopsis

text **PostGIS_Raster_Lib_Build_Date();**

安装

PostGIS_Raster_Lib_Build_Date。

安装

```
SELECT PostGIS_Raster_Lib_Build_Date();
       postgis_raster_lib_build_date
-----
2010-04-28 21:15:10
```

安装

`PostGIS_Raster_Lib_Version`

11.2.7 PostGIS_Raster_Lib_Version

PostGIS_Raster_Lib_Version — 返回 PostGIS 的 Raster 库的版本。

Synopsis

text **PostGIS_Raster_Lib_Version();**

安装

安装PostGIS 3.6.1。

安装

```
SELECT PostGIS_Raster_Lib_Version();
postgis_raster_lib_version
-----
2.0.0
```

安装

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);

安装

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.

安装: 2.0.6, 2.1.3 安装 - GUC 安装 gdal_enabled_drivers 安装, 安装

2.0.0 安装。 GDAL 1.6.0 安装。

安装: 安装

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SELECT short_name, long_name, can_write
FROM st_gdaldrivers()
ORDER BY short_name;
```

short_name	long_name	can_write
AAIGrid	Arc/Info ASCII Grid	t
ACE2	ACE2	f
ADRG	ARC Digitized Raster Graphics	f

AIG	Arc/Info Binary Grid	f
AirSAR	AirSAR Polarimetric Image	f
ARG	Azavea Raster Grid format	t
BAG	Bathymetry Attributed Grid	f
BIGGIF	Graphics Interchange Format (.gif)	f
BLX	Magellan topo (.blx)	t
BMP	MS Windows Device Independent Bitmap	f
BSB	Maptech BSB Nautical Charts	f
PAux	PCI .aux Labelled	f
PCIDSK	PCIDSK Database File	f
PCRaster	PCRaster Raster File	f
PDF	Geospatial PDF	f
PDS	NASA Planetary Data System	f
PDS4	NASA Planetary Data System 4	t
PLMOSAIC	Planet Labs Mosaics API	f
PLSCENES	Planet Labs Scenes API	f
PNG	Portable Network Graphics	t
PNM	Portable Pixmap Format (netpbm)	f
PRF	Racurs PHOTOMOD PRF	f
R	R Object Data Store	t
Rasterlite	Rasterlite	t
RDA	DigitalGlobe Raster Data Access driver	f
RIK	Swedish Grid RIK (.rik)	f
RMF	Raster Matrix Format	f
ROI_PAC	ROI_PAC raster	f
RPFTOC	Raster Product Format TOC format	f
RRASTER	R Raster	f
RS2	RadarSat 2 XML Product	f
RST	Idrisi Raster A.1	t
SAFE	Sentinel-1 SAR SAFE Product	f
SAGA	SAGA GIS Binary Grid (.sdat, .sg-grd-z)	t
SAR_CEOS	CEOS SAR Image	f
SDTS	SDTS Raster	f
SENTINEL2	Sentinel 2	f
SGI	SGI Image File Format 1.0	f
SNODAS	Snow Data Assimilation System	f
SRP	Standard Raster Product (ASRP/USRP)	f
SRTMHGT	SRTMHGT File Format	t
Terragen	Terragen heightfield	f
TIL	EarthWatch .TIL	f
TSX	TerraSAR-X Product	f
USGSDEM	USGS Optional ASCII DEM (and CDED)	t
VICAR	MIPL VICAR file	f
VRT	Virtual Raster	t
WCS	OGC Web Coverage Service	f
WMS	OGC Web Map Service	t
WMTS	OGC Web Map Tile Service	t
XPM	X11 PixMap Format	t
XYZ	ASCII Gridded XYZ	t
ZMap	ZMap Plus Grid	t

```
-- Output the create options XML column of JPEG as a table --
-- Note you can use these creator options in ST_AsGDALRaster options argument
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM   st_qdalldrivers())
```

```
WHERE short_name = 'JPEG') As g;
```

oname	otype	descrip
PROGRESSIVE	boolean	whether to generate a progressive JPEG
QUALITY	int	good=100, bad=0, default=75
WORLDFILE	boolean	whether to generate a worldfile
INTERNAL_MASK	boolean	whether to generate a validity mask
COMMENT	string	Comment
SOURCE_ICC_PROFILE	string	ICC profile encoded in Base64
EXIF_THUMBNAIL	boolean	whether to generate an EXIF thumbnail(overview). By default its max dimension will be 128
THUMBNAIL_WIDTH	int	Forced thumbnail width
THUMBNAIL_HEIGHT	int	Forced thumbnail height

(9 rows)

```
-- raw xml output for creator options for GeoTiff --
```

```
SELECT create_options
```

```
FROM st_gdaldrivers()
```

```
WHERE short_name = 'GTiff';
```

```
<CreationOptionList>
```

```
  <Option name="COMPRESS" type="string-select">
```

```
    <Value
```

```
>NONE</Value>
```

```
    <Value
```

```
>LZW</Value>
```

```
    <Value
```

```
>PACKBITS</Value>
```

```
    <Value
```

```
>JPEG</Value>
```

```
    <Value
```

```
>CCITTRLE</Value>
```

```
    <Value
```

```
>CCITTFAX3</Value>
```

```
    <Value
```

```
>CCITTFAX4</Value>
```

```
    <Value
```

```
>DEFLATE</Value>
```

```
  </Option>
```

```
  <Option name="PREDICTOR" type="int" description="Predictor Type"/>
```

```
  <Option name="JPEG_QUALITY" type="int" description="JPEG quality 1-100" default="75"/>
```

```
  <Option name="ZLEVEL" type="int" description="DEFLATE compression level 1-9" default ←  
    ="6"/>
```

```
  <Option name="NBITS" type="int" description="BITS for sub-byte files (1-7), sub-uint16 ←  
    (9-15), sub-uint32 (17-31)"/>
```

```
  <Option name="INTERLEAVE" type="string-select" default="PIXEL">
```

```
    <Value
```

```
>BAND</Value>
```

```
    <Value
```

```
>PIXEL</Value>
```

```
  </Option>
```

```
  <Option name="TILED" type="boolean" description="Switch to tiled format"/>
```

```
  <Option name="TFW" type="boolean" description="Write out world file"/>
```

```
  <Option name="RPB" type="boolean" description="Write out .RPB (RPC) file"/>
```

```
  <Option name="BLOCKXSIZE" type="int" description="Tile Width"/>
```

```
  <Option name="BLOCKYSIZE" type="int" description="Tile/Strip Height"/>
```

```
  <Option name="PHOTOMETRIC" type="string-select">
```

```
    <Value
```

```
>MINISBLACK</Value>
```

```
    <Value
```

```

>MINISWHITE</Value>
  <Value
>PALETTE</Value>
  <Value
>RGB</Value>
  <Value
>CMYK</Value>
  <Value
>YCBCR</Value>
  <Value
>CIELAB</Value>
  <Value
>ICCLAB</Value>
  <Value
>ITULAB</Value>
  </Option>
  <Option name="SPARSE_OK" type="boolean" description="Can newly created files have ↵
    missing blocks?" default="FALSE"/>
  <Option name="ALPHA" type="boolean" description="Mark first extrasample as being alpha ↵
    "/>
  <Option name="PROFILE" type="string-select" default="GDALGeoTIFF">
    <Value
>GDALGeoTIFF</Value>
    <Value
>GeoTIFF</Value>
    <Value
>BASELINE</Value>
  </Option>
  <Option name="PIXELTYPE" type="string-select">
    <Value
>DEFAULT</Value>
    <Value
>SIGNEDBYTE</Value>
  </Option>
  <Option name="BIGTIFF" type="string-select" description="Force creation of BigTIFF file ↵
    ">
    <Value
>YES</Value>
    <Value
>NO</Value>
    <Value
>IF_NEEDED</Value>
    <Value
>IF_SAFER</Value>
  </Option>
  <Option name="ENDIANNESS" type="string-select" default="NATIVE" description="Force ↵
    endianness of created file. For DEBUG purpose mostly">
    <Value
>NATIVE</Value>
    <Value
>INVERTED</Value>
    <Value
>LITTLE</Value>
    <Value
>BIG</Value>
  </Option>
  <Option name="COPY_SRC_OVERVIEWS" type="boolean" default="NO" description="Force copy ↵
    of overviews of source dataset (CreateCopy())"/>
</CreationOptionList>

-- Output the create options XML column for GTiff as a table --
SELECT (xpath('@name', g.opt))[1]::text As oname,

```

```

        (xpath('@type', g.opt))[1]::text As otype,
        (xpath('@description', g.opt))[1]::text As descrip,
        array_to_string(xpath('Value/text()', g.opt),', ') As vals
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM st_gdaldrivers()
WHERE short_name = 'GTiff') As g;

```

oname	otype	descrip	vals
COMPRESS	string-select		NONE, LZW, ←
PACKBITS, JPEG, CCITTRLE, CCITTFAX3, CCITTFAX4, DEFLATE			
PREDICTOR	int	Predictor Type	←
JPEG_QUALITY	int	JPEG quality 1-100	←
ZLEVEL	int	DEFLATE compression level 1-9	←
NBITS	int	BITS for sub-byte files (1-7), sub-uint16 (9-15), sub-uint32 (17-31)	←
INTERLEAVE	string-select		BAND, PIXEL
TILED	boolean	Switch to tiled format	←
TFW	boolean	Write out world file	←
RPB	boolean	Write out .RPB (RPC) file	←
BLOCKXSIZE	int	Tile Width	←
BLOCKYSIZE	int	Tile/Strip Height	←
PHOTOMETRIC	string-select		MINISBLACK, ←
MINISWHITE, PALETTE, RGB, CMYK, YCBCR, CIELAB, ICCLAB, ITULAB			
SPARSE_OK	boolean	Can newly created files have missing blocks?	←
ALPHA	boolean	Mark first extrasample as being alpha	←
PROFILE	string-select		GDALGeoTIFF, ←
GeoTIFF, BASELINE			
PIXELTYPE	string-select		DEFAULT, ←
SIGNEDBYTE			
BIGTIFF	string-select	Force creation of BigTIFF file	←
ENDIANNESS	string-select	Force endianness of created file. For DEBUG purpose	←
mostly			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 — 指定した SRID に更新する。

Synopsis

```
raster UpdateRasterSRID(name schema_name, name table_name, name column_name, integer new_srid);
raster UpdateRasterSRID(name table_name, name column_name, integer new_srid);
```

更新

指定した SRID に更新する。指定した SRID が指定された (列, 表) の SRID に更新される。



Note

指定した SRID (列, 表) が指定された SRID に更新される。

2.1.0 更新内容。

更新

UpdateGeometrySRID

11.2.10 ST_CreateOverview

ST_CreateOverview — 指定した SRID に更新する。

Synopsis

```
regclass ST_CreateOverview(regclass tab, name col, int factor, text algo='NearestNeighbor');
```

更新

指定した SRID に更新する。指定した SRID が指定された (列, 表) の SRID に更新される。

raster_overviews 指定した SRID に更新する。

指定した SRID に更新する。指定した SRID が指定された (列, 表) の SRID に更新される。指定した SRID が指定された (列, 表) の SRID に更新される。

2.2.0 更新内容。

更新

Output to generally better quality but slower to product format

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2, 'Lanczos');
```

Output to faster to process default nearest neighbor

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2);
```


ST_Retile, AddOverviewConstraints, AddRasterConstraints, Section 10.2.2

11.3 (constructor)

11.3.1 ST_AddBand

ST AddBand —       () .   ,   .

Synopsis

- (1) raster **ST_AddBand**(raster rast, addbandarg[] addbandargset);
- (2) raster **ST_AddBand**(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
- (3) raster **ST_AddBand**(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
- (4) raster **ST_AddBand**(raster torast, raster fromrast, integer fromband=1, integer torastindex=at_end);
- (5) raster **ST_AddBand**(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at_end);
- (6) raster **ST_AddBand**(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);
- (7) raster **ST_AddBand**(raster rast, text outdbfile, integer[] outdbindex, integer index=at_end, double precision nodataval=NULL);

Returns a raster with a new band added in given position (index), of given type, of given initial value, and of given nodata value. If no index is specified, the band is added to the end. If no fromband is specified, band 1 is assumed. Pixel type is a string representation of one of the pixel types specified in [ST_BandPixelType](#). If an existing index is specified all subsequent bands $\geq$ that index are incremented by 1. If an initial value greater than the max of the pixel type is specified, then the initial value is set to the highest value allowed by the pixel type.

[illegible]

XXXXXXXXXXXXXXXXXXXX 5 XXXX, torast X NULL XXXXXXXXXXXXXXXXXXXX fromband XXXXXXXX
XXXXXX (累計) XXXX.

outdbfile 6 7 , outdbfile .
PostgreSQL .

2.1.0 addbandarg 2.1.0.

MySQL: 2.1.0 DB MySQL.

[illegible]

```
-- Add another band of type 8 bit unsigned integer with pixels initialized to 200
UPDATE dummy_rast
    SET rast = ST_AddBand(rast,'8BUI'::text,200)
WHERE rid = 1;
```

```
-- Create an empty raster 100x100 units, with upper left right at 0, add 2 bands (band 1 ←
is 0/1 boolean bit switch, band2 allows values 0-15)
-- uses addbandargs
INSERT INTO dummy_rast(rid,rast)
VALUES(10, ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 1, -1, 0, 0, 0),
ARRAY[
    ROW(1, '1BB'::text, 0, NULL),
    ROW(2, '4BUI'::text, 0, NULL)
]::addbandarg[]
)
);

-- output meta data of raster bands to verify all is right --
SELECT (bmd).*
FROM (SELECT ST_BandMetaData(rast,generate_series(1,2)) As bmd
FROM dummy_rast WHERE rid = 10) AS foo;
--result --
pixeltype | nodatavalue | isoutdb | path
-----+-----+-----+-----
1BB       |             | f       |
4BUI      |             | f       |

-- output meta data of raster -
SELECT (rmd).width, (rmd).height, (rmd).numbands
FROM (SELECT ST_MetaData(rast) As rmd
FROM dummy_rast WHERE rid = 10) AS foo;
-- result --
upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | ←
numbands
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | 0 | 100 | 100 | 1 | -1 | 0 | 0 | 0 | ←
2
```

例: 创建带元数据的栅格

```
SELECT
*
FROM ST_BandMetadata(
    ST_AddBand(
        ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
        ARRAY[
            ROW(NULL, '8BUI', 255, 0),
            ROW(NULL, '16BUI', 1, 2),
            ROW(2, '32BUI', 100, 12),
            ROW(2, '32BF', 3.14, -1)
        ]::addbandarg[]
    ),
    ARRAY[]::integer[]
);

bandnum | pixeltype | nodatavalue | isoutdb | path
-----+-----+-----+-----+-----
1 | 8BUI      | 0           | f       |
2 | 32BF      | -1          | f       |
3 | 32BUI     | 12          | f       |
4 | 16BUI     | 2           | f       |
```

```
-- Aggregate the 1st band of a table of like rasters into a single raster
-- with as many bands as there are test_types and as many rows (new rasters) as there are ←
-- mice
-- NOTE: The ORDER BY test_type is only supported in PostgreSQL 9.0+
-- for 8.4 and below it usually works to order your data in a subselect (but not guaranteed ←
-- )
-- The resulting raster will have a band for each test_type alphabetical by test_type
-- For mouse lovers: No mice were harmed in this exercise
SELECT
  mouse,
  ST_AddBand(NULL, array_agg(rast ORDER BY test_type), 1) As rast
FROM mice_studies
GROUP BY mouse;
```

图 11.3.2: 使用 DB 函数

```
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

图 11.3.3

[ST_BandMetaData](#), [ST_BandPixelType](#), [ST_MakeEmptyRaster](#), [ST_MetaData](#), [ST_NumBands](#), [ST_Reclass](#)

11.3.2 ST_AsRaster

ST_AsRaster — PostGIS 函数 PostGIS 函数

Synopsis

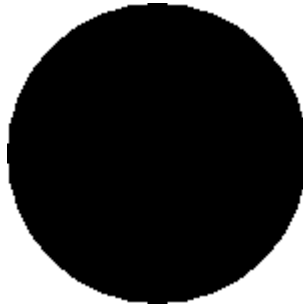
raster **ST_AsRaster**(geometry geom, raster ref, text pixeltype, double precision value=1, double precision nodataval=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, raster ref, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);

2.0.0 00000000000000000000. GDAL 1.6.0 00000000000000000000.

**Note**

ST_AsRaster, TIN, ST_GeomFromTIN, ST_GeomFromWKB, GDAL 驱动程序。ST_AsRaster 使用 GDAL 驱动程序。

图例: PNG 格式输出



图例

```
-- this will output a black circle taking up 150 x 150 pixels --
SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10),150, 150));
```



PostGIS 图例

```
-- the bands map to RGB bands - the value (118,154,118) - teal --
SELECT ST_AsPNG(
  ST_AsRaster(
    ST_Buffer(
      ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10,'join=bevel'),
      200,200,ARRAY['8BUI', '8BUI', '8BUI'], ARRAY[118,154,118], ARRAY[0,0,0]));
```

图例

[ST_BandPixelType](#), [ST_Buffer](#), [ST_GDALDrivers](#), [ST_AsGDALRaster](#), [ST_AsPNG](#), [ST_AsJPEG](#), [ST_SRID](#)

11.3.3 ST_AsRasterAgg

ST_AsRasterAgg — Aggregate. Renders PostGIS geometries into a new raster.

Synopsis

raster **ST_AsRasterAgg**(geometry geom, double precision val, raster ref, text pixeltype, double precision nodataval, text uniontype, boolean touched);

返回

Returns a single-band raster containing the rendered version of all incoming geometries, each with its associated value.

Availability: 3.6.0

示例

```
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)

参见

[ST_AsRaster](#), [ST_DumpAsPolygons](#), [ST_Union](#)

11.3.4 ST_Band

ST_Band — 返回指定栅格的波段索引。返回的索引从 1 开始，按波段在栅格中的存储顺序排列。如果栅格是单波段的，则返回 1。

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=,);
```

返回

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 是空数组，则返回所有带。使用 '1,2,3' 指定带时，ST_Band(rast, '1@2@3', '@') 返回所有带。ST_Band(rast, '{1,2,3}'::int[]) 返回指定的带。PostGIS 3.6.0 之前，text 类型的 nbands 参数会被忽略。

2.0.0 版本中，nbands 参数是必需的。

示例

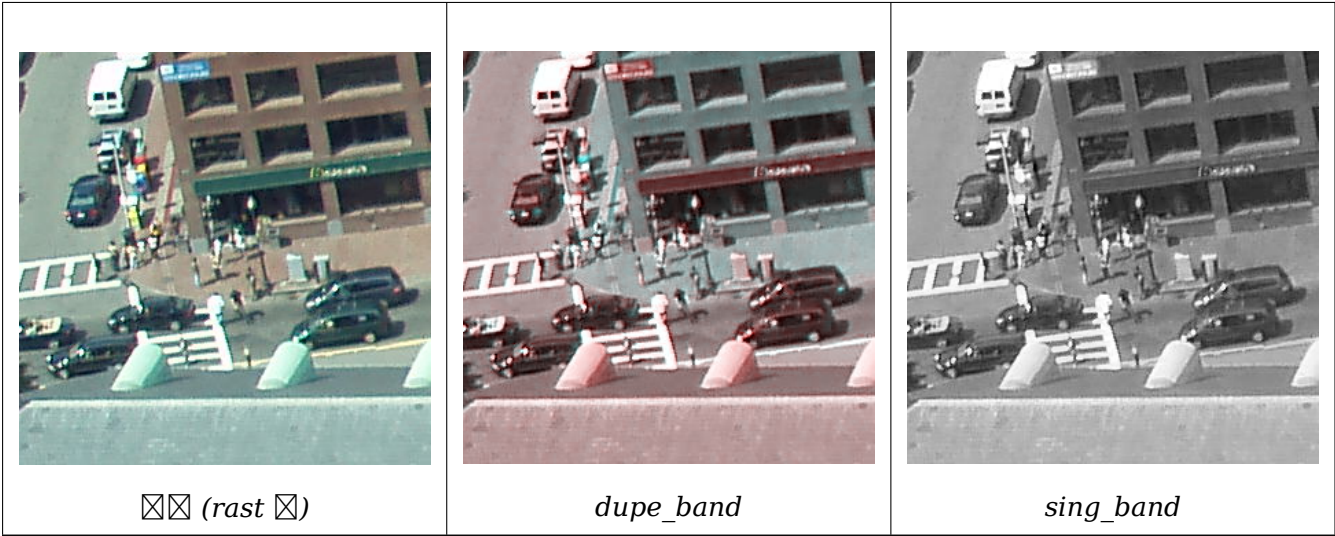
```
-- Make 2 new rasters: 1 containing band 1 of dummy, second containing band 2 of dummy and then reclassified as a 2BUI
SELECT ST_NumBands(rast1) As numb1, ST_BandPixelType(rast1) As pix1,
       ST_NumBands(rast2) As numb2, ST_BandPixelType(rast2) As pix2
FROM (
  SELECT ST_Band(rast) As rast1, ST_Reclass(ST_Band(rast,3), '100-200):1, [200-254:2', '2BUI') As rast2
  FROM dummy_rast
  WHERE rid = 2) As foo;
```

numb1	pix1	numb2	pix2
1	8BUI	1	2BUI

```
-- Return bands 2 and 3. Using array cast syntax
SELECT ST_NumBands(ST_Band(rast, '{2,3}'::int[])) As num_bands
FROM dummy_rast WHERE rid=2;
```

num_bands
2

```
-- Return bands 2 and 3. Use array to define bands
SELECT ST_NumBands(ST_Band(rast, ARRAY[2,3])) As num_bands
FROM dummy_rast
WHERE rid=2;
```



```
--Make a new raster with 2nd band of original and 1st band repeated twice,  
and another with just the third band  
SELECT rast, ST_Band(rast, ARRAY[2,1,1]) As dupe_band,  
       ST_Band(rast, 3) As sing_band  
FROM samples.than_chunked  
WHERE rid=35;
```

栅格

[ST_AddBand](#), [ST_NumBands](#), [ST_Reclass](#), Chapter 11

11.3.5 ST_MakeEmptyCoverage

ST_MakeEmptyCoverage — Cover georeferenced area with a grid of empty raster tiles.

Synopsis

raster **ST_MakeEmptyCoverage**(integer tilewidth, integer tileheight, integer width, integer height, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy, integer srid=unknown);

栅格

Create a set of raster tiles with [ST_MakeEmptyRaster](#). Grid dimension is width & height. Tile dimension is tilewidth & tileheight. The covered georeferenced area is from upper left corner (upperleftx, upperlefty) to lower right corner (upperleftx + width * scalex, upperlefty + height * scaley).



Note
Note that scaley is generally negative for rasters and scalex is generally positive. So lower right corner will have a lower y value and higher x value than the upper left corner.

示例

Create 16 tiles in a 4x4 grid to cover the WGS84 area from upper left corner (22, 77) to lower right corner (55, 33).

```
SELECT (ST_MetaData(tile)).* FROM ST_MakeEmptyCoverage(1, 1, 4, 4, 22, 33, (55 - 22)/(4)::float, (33 - 77)/(4)::float, 0., 0., 4326) tile;
```

upperleftx numbands	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	
22	33	1	1	8.25	-11	0	0	4326	
30.25	0	33	1	8.25	-11	0	0	4326	
38.5	0	33	1	8.25	-11	0	0	4326	
46.75	0	33	1	8.25	-11	0	0	4326	
22	22	1	1	8.25	-11	0	0	4326	
30.25	0	22	1	8.25	-11	0	0	4326	
38.5	0	22	1	8.25	-11	0	0	4326	
46.75	0	22	1	8.25	-11	0	0	4326	
22	11	1	1	8.25	-11	0	0	4326	
30.25	0	11	1	8.25	-11	0	0	4326	
38.5	0	11	1	8.25	-11	0	0	4326	
46.75	0	11	1	8.25	-11	0	0	4326	
22	0	0	1	8.25	-11	0	0	4326	
30.25	0	0	1	8.25	-11	0	0	4326	
38.5	0	0	1	8.25	-11	0	0	4326	
46.75	0	0	1	8.25	-11	0	0	4326	

函数

ST_MakeEmptyRaster

11.3.6 ST_MakeEmptyRaster

ST_MakeEmptyRaster — 创建空栅格 (列 & 行), 列 X 行, 比例尺, 偏斜 (scalex, scaley, skewx & skewy) 和 SRID (SRID) 的栅格。 列, 行, 比例尺, 偏斜 SRID 是可选的。 SRID 未指定, 默认为 0(unknown)。

Synopsis

```
raster ST_MakeEmptyRaster(raster rast);
raster ST_MakeEmptyRaster(integer width, integer height, float8 upperleftx, float8 upperlefty, float8
scalex, float8 scaley, float8 skewx, float8 skewy, integer srid=unknown);
raster ST_MakeEmptyRaster(integer width, integer height, float8 upperleftx, float8 upperlefty, float8
pixelsize);
```



00000 (00 & 00), 00 (0000) 000000000000 X(upperleftx) 0000 Y(upperlefty), 00
 000, 00 (scalex, scaley, skewx & skewy) 000000000000 (SRID) 00000 (00000) 000
 0000000.

00000000 (pixelsize) 00000000000000000000. scalex 00000000,
 scaley 0000000000000000. skewx 0 skewy 0 00000000.

□□□□□□□□□□, □□□□□□□□□□□□□□□□ (□□□□□) □□□□□□□□□□.

SRID 0. 0. ST AddBand, ST SetValue.


```
INSERT INTO dummy_rast(rid,rast)
VALUES(3, ST_MakeEmptyRaster( 100, 100, 0.0005, 0.0005, 1, 1, 0, 0, 4326) );
```

```
--use an existing raster as template for new raster
INSERT INTO dummy_rast(rid,rast)
SELECT 4, ST_MakeEmptyRaster(rast)
FROM dummy_rast WHERE rid = 3;
```

```
-- output meta data of rasters we just added
SELECT rid, (md).*
FROM (SELECT rid, ST_MetaData(rast) As md
      FROM dummy_rast
      WHERE rid IN(3,4)) As foo;
```

```
-- output --
rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | ↵
      numbands
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
  3 |    0.0005 |    0.0005 |   100 |   100 |      1 |      1 |      1 |      0 |      0 | ↵
    4326 |          |  0
  4 |    0.0005 |    0.0005 |   100 |   100 |      1 |      1 |      1 |      0 |      0 | ↵
    4326 |          |  0
```



ST AddBand, ST MetaData, ST ScaleX, ST ScaleY, ST SetValue, ST SkewX, , ST SkewY

11.3.7 ST Tile

ST Tile — 

Synopsis

setof raster **ST_Tile**(raster rast, int[] nband, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);
 setof raster **ST_Tile**(raster rast, integer nband, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);
 setof raster **ST_Tile**(raster rast, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);

参数

返回类型: 栅格集合 (set of raster)。

padwithnodata = FALSE 时, 如果栅格集合中的栅格具有不同的带数, 则使用 NODATA 值填充 (padding) 缺失的带。如果 padwithnodata = TRUE, 则使用 NODATA 值填充缺失的带。nodataval 指定 NODATA 值。



Note

如果栅格集合中的栅格具有不同的带数, 则使用 NODATA 值填充缺失的带。

2.1.0 版本更新

参数

```
WITH foo AS (
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', 1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', 2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', 3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
  SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
  SELECT ST_Tile(rast, 3, 3, TRUE) AS rast FROM bar
)
SELECT
  ST_DumpValues(rast)
FROM baz;
```


st_dumpvalues

```
-----
(1,"{{1,1,1},{1,1,1},{1,1,1}}")
(2,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{2,2,2},{2,2,2},{2,2,2}}")
(2,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{3,3,3},{3,3,3},{3,3,3}}")
(2,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{4,4,4},{4,4,4},{4,4,4}}")
(2,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{5,5,5},{5,5,5},{5,5,5}}")
(2,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{6,6,6},{6,6,6},{6,6,6}}")
(2,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{7,7,7},{7,7,7},{7,7,7}}")
(2,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{8,8,8},{8,8,8},{8,8,8}}")
(2,"{{80,80,80},{80,80,80},{80,80,80}}")
(1,"{{9,9,9},{9,9,9},{9,9,9}}")
(2,"{{90,90,90},{90,90,90},{90,90,90}}")
(18 rows)
```

```
WITH foo AS (
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', 1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', 2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', 3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
  SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
  SELECT ST_Tile(rast, 3, 3, 2) AS rast FROM bar
)
SELECT
  ST_DumpValues(rast)
FROM baz;
```

st_dumpvalues

```
-----
(1,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{80,80,80},{80,80,80},{80,80,80}}")
```

```
(1,"{{90,90,90},{90,90,90},{90,90,90}}")
(9 rows)
```

☐☐

[ST_Union](#), [ST_Retile](#)

11.3.8 ST_Retile

`ST_Retile` — 将栅格重采样为指定的宽度和高度。

Synopsis

SETOF raster **ST_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

☐☐

`sfx`, `sfy` 指定栅格的缩放因子，`tw`, `th` 指定栅格的宽度和高度，`tab`, `col` 指定栅格的表和列，`ext` 指定栅格的几何类型。

支持的 `algo` 值包括 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', 以及 'Lanczos'。☐☐ [GDAL Warp resampling methods](#) 文档。

2.2.0 版本引入。

☐☐

[ST_CreateOverview](#)

11.3.9 ST_FromGDALRaster

`ST_FromGDALRaster` — 从 GDAL 数据集创建栅格。

Synopsis

raster **ST_FromGDALRaster**(bytea gdaldata, integer srid=NULL);

☐☐

`gdaldata` 是 GDAL 数据集的字节数据。☐☐ GDAL 数据集的 SRID 值。☐☐ `srid` 是 SRID 的整数表示。

如果 `srid` 为 NULL，则使用 GDAL 数据集的 SRID。☐☐ `srid` 是 SRID 的整数表示。

2.1.0 版本引入。

例

```
WITH foo AS (
  SELECT ST_AsPNG(ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.1, ←
    -0.1, 0, 0, 4326), 1, '8BUI', 1, 0), 2, '8BUI', 2, 0), 3, '8BUI', 3, 0)) AS png
),
bar AS (
  SELECT 1 AS rid, ST_FromGDALRaster(png) AS rast FROM foo
  UNION ALL
  SELECT 2 AS rid, ST_FromGDALRaster(png, 3310) AS rast FROM foo
)
SELECT
  rid,
  ST_Metadata(rast) AS metadata,
  ST_SummaryStats(rast, 1) AS stats1,
  ST_SummaryStats(rast, 2) AS stats2,
  ST_SummaryStats(rast, 3) AS stats3
FROM bar
ORDER BY rid;
```

rid	metadata	stats1	stats2	stats3
1	(0,0,2,2,1,-1,0,0,0,3)	(4,4,1,0,1,1)	(4,8,2,0,2,2)	(4,12,3,0,3,3)
2	(0,0,2,2,1,-1,0,0,3310,3)	(4,4,1,0,1,1)	(4,8,2,0,2,2)	(4,12,3,0,3,3)

(2 rows)

例

ST_AsGDALRaster

11.4 函数 (accessor)

11.4.1 ST_GeoReference

ST_GeoReference — 返回 (world) 坐标参考系信息 GDAL 或 ESRI 格式。

Synopsis

text **ST_GeoReference**(raster rast, text format=GDAL);

例

例 1 返回 GDAL 格式。例 2 返回 ESRI 格式。

例 3 返回 GDAL 或 ESRI 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

返回栅格的垂直高度。

例

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;
```

rid	rastheight
1	20
2	5

函数

ST_Width

11.4.3 ST_IsEmpty

ST_IsEmpty — 检查空栅格 (width = 0, height = 0) 是否有效。返回布尔值。

Synopsis

boolean **ST_IsEmpty**(raster rast);

函数

检查空栅格 (width = 0, height = 0) 是否有效。返回布尔值。
2.0.0 版本引入。

函数

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0))
st_isempty |
-----+
f          |

SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0))
st_isempty |
-----+
t          |
```

函数

ST_HasNoBand

11.4.4 ST_MemSize

ST_MemSize — 返回栅格在内存中的大小 (字节)。

Synopsis

integer **ST_MemSize**(raster rast);

1		0.5		0.5		10		20		2		3		0		0		0		↵
2		3427927.75		5793244		5		5		0.05		-0.05		0		0		0		↵
		0																		
		3																		

返回

[ST_BandMetaData](#), [ST_NumBands](#)

11.4.6 ST_NumBands

ST_NumBands — 返回栅格中的波段数。

Synopsis

integer **ST_NumBands**(raster rast);

返回

返回栅格中的波段数。

返回

```
SELECT rid, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

rid	numbands
1	0
2	3

返回

[ST_Value](#)

11.4.7 ST_PixelHeight

ST_PixelHeight — 返回栅格中的像素高度。

Synopsis

double precision **ST_PixelHeight**(raster rast);

返回

返回栅格中的像素高度。如果栅格是二维的，则返回像素高度。如果栅格是三维的，则返回像素高度和像素宽度的乘积。

返回栅格中的像素高度。如果栅格是二维的，则返回像素高度。如果栅格是三维的，则返回像素高度和像素宽度的乘积。 [ST_PixelWidth](#) 返回像素宽度。

例: 返回栅格属性

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM dummy_rast;
```

rastheight	pixheight	scalex	scaley	skewx	skewy
20	3	2	3	0	0
5	0.05	0.05	-0.05	0	0

例: 0 度旋转栅格

```
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。

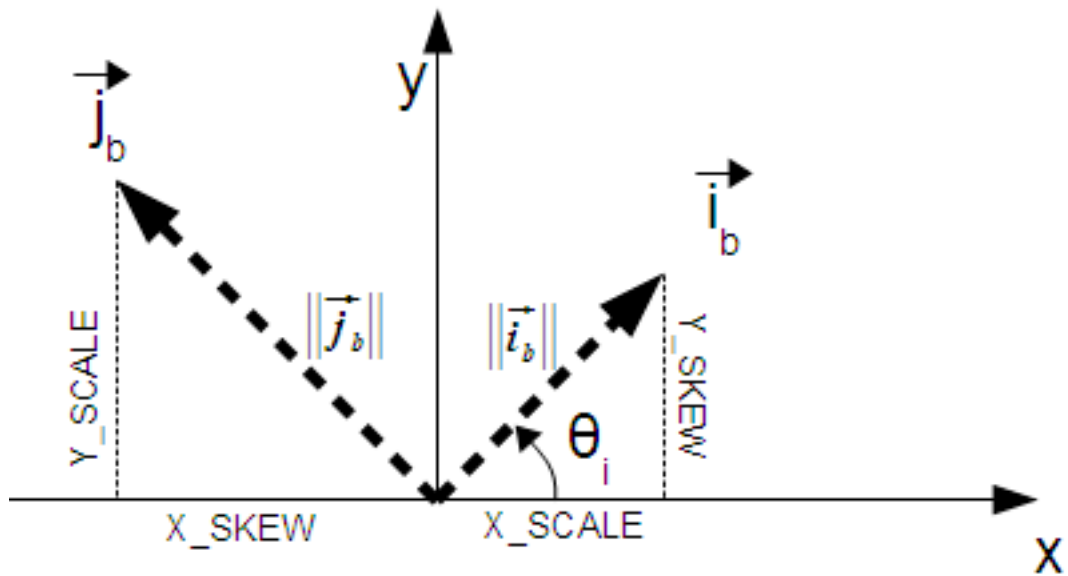
Synopsis

double precision **ST_PixelWidth**(raster rast);

例

返回栅格的像素宽度。对于非正方形栅格，返回的像素宽度与像素高度不同。

对于正方形栅格，返回的像素宽度与像素高度相同。



i
 j

图 1: 坐标系统

```
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 2: 0 坐标系统

```
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 3

[ST_PixelHeight](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SkewX](#), [ST_SkewY](#)

11.4.9 ST_ScaleX

ST_ScaleX — 将栅格的 X 轴像素大小缩放为指定的值。

Synopsis

float8 **ST_ScaleX**(raster rast);

返回

返回栅格的 X 轴像素大小。如果指定的值不为 1，则返回的像素大小将乘以指定的缩放因子。
版本: 2.0.0 在 WKTRaster 中通过 ST_PixelSizeX 引入。

示例

```
SELECT rid, ST_ScaleX(rast) As rastpixwidth
FROM dummy_rast;
```

rid	rastpixwidth
1	2
2	0.05

注意

ST_Width

11.4.10 ST_ScaleY

ST_ScaleY — 将栅格的 Y 轴像素大小缩放为指定的值。

Synopsis

float8 **ST_ScaleY**(raster rast);

返回

返回栅格的 Y 轴像素大小。如果指定的值不为 1，则返回的像素大小将乘以指定的缩放因子。
版本: 2.0.0 在 WKTRaster 中通过 ST_PixelSizeY 引入。

示例

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;
```

rid	rastpixheight
1	3
2	-0.05

函数

ST_Height

11.4.11 ST_RasterToWorldCoord

ST_RasterToWorldCoord — 返回栅格像素 (x, y) 的地理坐标。
返回类型为 geometry 或 geography，取决于输入栅格的类型。

Synopsis

record **ST_RasterToWorldCoord**(raster rast, integer xcolumn, integer yrow);

函数

返回栅格像素 (x, y) 的地理坐标。x, y 是栅格像素的列索引和行索引。索引从 0 开始。如果栅格是 skewed，则返回的坐标是 skewed 后的坐标。

2.1.0 版本引入。

函数

```
-- non-skewed raster
SELECT
  rid,
  (ST_RasterToWorldCoord(rast,1, 1)).*,
  (ST_RasterToWorldCoord(rast,2, 2)).*
FROM dummy_rast
```

rid	longitude	latitude	longitude	latitude
1	0.5	0.5	2.5	3.5
2	3427927.75	5793244	3427927.8	5793243.95

```
-- skewed raster
SELECT
  rid,
  (ST_RasterToWorldCoord(rast, 1, 1)).*,
  (ST_RasterToWorldCoord(rast, 2, 3)).*
FROM (
  SELECT
    rid,
    ST_SetSkew(rast, 100.5, 0) As rast
  FROM dummy_rast
) As foo
```

rid	longitude	latitude	longitude	latitude
1	0.5	0.5	203.5	6.5
2	3427927.75	5793244	3428128.8	5793243.9

函数

[ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SetSkew](#)

11.4.12 ST_RasterToWorldCoordX

`ST_RasterToWorldCoordX` — 给定栅格和列索引，返回该列的 X 世界坐标。返回类型是 `float8`。

Synopsis

```
float8 ST_RasterToWorldCoordX(raster rast, integer xcolumn);
float8 ST_RasterToWorldCoordX(raster rast, integer xcolumn, integer yrow);
```

函数

给定栅格和列索引，返回该列的 X 世界坐标。返回类型是 `float8`。如果提供了行索引，则返回该像素的世界坐标。否则，返回该列的 X 世界坐标。



Note

如果提供了行索引，则返回该像素的世界坐标。否则，返回该列的 X 世界坐标。使用 `ST_ScaleX`, `ST_SkewX`, 和 `ST_SetSkew` 函数可以设置栅格的 X 世界坐标。

从 2.1.0 版本开始，`ST_Raster2WorldCoordX` 函数已弃用。

函数

```
-- non-skewed raster providing column is sufficient
SELECT rid, ST_RasterToWorldCoordX(rast,1) As xlcoord,
       ST_RasterToWorldCoordX(rast,2) As x2coord,
       ST_ScaleX(rast) As pixelx
FROM dummy_rast;
```

rid	xlcoord	x2coord	pixelx
1	0.5	2.5	2
2	3427927.75	3427927.8	0.05

```
-- for fun lets skew it
SELECT rid, ST_RasterToWorldCoordX(rast, 1, 1) As xlcoord,
       ST_RasterToWorldCoordX(rast, 2, 3) As x2coord,
       ST_ScaleX(rast) As pixelx
FROM (SELECT rid, ST_SetSkew(rast, 100.5, 0) As rast FROM dummy_rast) As foo;
```

rid	xlcoord	x2coord	pixelx
1	0.5	203.5	2
2	3427927.75	3428128.8	0.05

函数

[ST_ScaleX](#), [ST_RasterToWorldCoordY](#), [ST_SetSkew](#), [ST_SkewX](#)

11.4.13 ST_RasterToWorldCoordY

`ST_RasterToWorldCoordY` — 返回栅格中指定行 Y 的地理坐标。返回类型为 `float8`。

Synopsis

```
float8 ST_RasterToWorldCoordY(raster rast, integer yrow);
float8 ST_RasterToWorldCoordY(raster rast, integer xcolumn, integer yrow);
```

函数

`ST_RasterToWorldCoordY` 返回栅格中指定行 Y 的地理坐标。返回类型为 `float8`。如果栅格是未偏斜的，那么只需要提供行号。如果栅格是偏斜的，那么还需要提供列号。返回的坐标是相对于栅格的左上角（0,0）的。



Note

如果栅格是偏斜的，Y 坐标是相对于栅格的左上角（0,0）的。如果栅格是未偏斜的，那么只需要提供行号。返回的坐标是相对于栅格的左上角（0,0）的。如果栅格是偏斜的，那么还需要提供列号。返回的坐标是相对于栅格的左上角（0,0）的。

注意：2.1.0 版本中 `ST_Raster2WorldCoordY` 函数已弃用。

函数

```
-- non-skewed raster providing row is sufficient
SELECT rid, ST_RasterToWorldCoordY(rast,1) As ylcoord,
       ST_RasterToWorldCoordY(rast,3) As y2coord,
       ST_ScaleY(rast) As pixely
FROM dummy_rast;
```

rid	ylcoord	y2coord	pixely
1	0.5	6.5	3
2	5793244	5793243.9	-0.05

```
-- for fun lets skew it
SELECT rid, ST_RasterToWorldCoordY(rast,1,1) As ylcoord,
       ST_RasterToWorldCoordY(rast,2,3) As y2coord,
       ST_ScaleY(rast) As pixely
FROM (SELECT rid, ST_SetSkew(rast,0,100.5) As rast FROM dummy_rast) As foo;
```

rid	ylcoord	y2coord	pixely
1	0.5	107	3
2	5793244	5793344.4	-0.05

[ST_ScaleY](#), [ST_RasterToWorldCoordX](#), [ST_SetSkew](#), [ST_SkewY](#)

11.4.14 ST_Rotation

ST_Rotation — 返回栅格的旋转角度。

Synopsis

float8 **ST_Rotation**(raster rast);

返回栅格的旋转角度。如果栅格没有旋转，则返回 0。如果栅格是 NaN，则返回 NaN。如果栅格是空集，则返回 0。

```
SELECT rid, ST_Rotation(ST_SetScale(ST_SetSkew(rast, sqrt(2)), sqrt(2))) as rot FROM dummy_rast;
```

rid	rot
1	0.785398163397448
2	0.785398163397448

[ST_SetRotation](#), [ST_SetScale](#), [ST_SetSkew](#)

11.4.15 ST_SkewX

ST_SkewX — 返回栅格的 X 轴偏斜 (skew) 角度。

Synopsis

float8 **ST_SkewX**(raster rast);

返回栅格的 X 轴偏斜 (skew) 角度。如果栅格没有偏斜，则返回 0。如果栅格是 NaN，则返回 NaN。如果栅格是空集，则返回 0。

例

```
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast;
```

rid	skewx	skewy	georef
1	0	0	2.0000000000
			: 0.0000000000
			: 0.0000000000
			: 3.0000000000
			: 0.5000000000
			: 0.5000000000
2	0	0	: 0.0500000000
			: 0.0000000000
			: 0.0000000000
			: -0.0500000000
			: 3427927.7500000000
			: 5793244.0000000000

例

ST_GeoReference, ST_SkewY, ST_SetSkew

11.4.16 ST_SkewY

ST_SkewY — 返回 Y 轴上的偏斜 (以度为单位) 的浮点值。

Synopsis

```
float8 ST_SkewY(raster rast);
```

例

返回 Y 轴上的偏斜 (以度为单位) 的浮点值。返回值为 **0.0000000000**。

例

```
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast;
```

rid	skewx	skewy	georef
1	0	0	2.0000000000
			: 0.0000000000
			: 0.0000000000
			: 3.0000000000
			: 0.5000000000
			: 0.5000000000



11.4.17 ST_SRID

Synopsis



Note!

PostGIS 2.0 4.4.1, 4.4.1/4.4.1 SRID 4326 -1 0 4.4.1/4.4.1



11.4.18 ST_Summary

Synopsis

```
text ST_Summary(raster rast);
```


图

[ST_UpperLeftY](#), [ST_GeoReference](#), [Box3D](#)

11.4.20 ST_UpperLeftY

`ST_UpperLeftY` — 返回栅格的左上角 Y 坐标。

Synopsis

`float8 ST_UpperLeftY(raster rast);`

图

返回栅格的左上角 Y 坐标。

图

```
SELECT rid, ST_UpperLeftY(rast) As uly
FROM dummy_rast;
```

rid	uly
1	0.5
2	5793244

图

[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);`

图

返回栅格的宽度。

例

```
SELECT ST_Width(rast) As rastwidth
FROM dummy_rast WHERE rid=1;

rastwidth
-----
10
```

例

ST_Height

11.4.22 ST_WorldToRasterCoord

ST_WorldToRasterCoord — 给定 X, Y(经度, 纬度) 返回栅格中的列和行索引。

Synopsis

record **ST_WorldToRasterCoord**(raster rast, geometry pt);
record **ST_WorldToRasterCoord**(raster rast, double precision longitude, double precision latitude);

例

给定 X, Y(经度, 纬度) 返回栅格中的列和行索引。给定 X, Y 返回栅格中的列和行索引。给定 X, Y 返回栅格中的列和行索引。

2.1.0 版本引入。

例

```
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
```

例

ST_WorldToRasterCoordX, **ST_WorldToRasterCoordY**, **ST_RasterToWorldCoordX**, **ST_RasterToWorldCoordY**, **ST_SRID**

11.4.23 ST_WorldToRasterCoordX

`ST_WorldToRasterCoordX` — 给定 (pt) 的 X 坐标 (xw, yw)。

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);
```

返回

给定 (pt) 的 X 坐标 (xw, yw)。如果 (xw, yw) 是 (xw, yw) 的坐标，则返回 xw。如果 (xw, yw) 是 (xw, yw) 的坐标，则返回 yw。如果 (xw, yw) 是 (xw, yw) 的坐标，则返回 xw。如果 (xw, yw) 是 (xw, yw) 的坐标，则返回 yw。

版本: 2.1.0 添加 `ST_World2RasterCoordX`。

示例

```
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

返回

`ST_RasterToWorldCoordX`, `ST_RasterToWorldCoordY`, `ST_SRID`

11.4.24 ST_WorldToRasterCoordY

`ST_WorldToRasterCoordY` — 给定 (pt) 的 Y 坐标 (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);
```




Note
If `isoutdb` is `False`, `path`, `outdbbandnum`, `filesize` and `filetimestamp` are `NULL`. If `outdb` access is disabled, `filesize` and `filetimestamp` will also be `NULL`.

Enhanced: 2.5.0 to include *outdbbandnum*, *filesize* and *filetimestamp* for `outdb` rasters.

Example 1

```
SELECT
  rid,
  (foo.md).*
FROM (
  SELECT
    rid,
    ST_BandMetaData(rast, 1) AS md
  FROM dummy_rast
  WHERE rid=2
) As foo;
```

rid	pixeltype	nodatavalue	isoutdb	path	outdbbandnum
2	8BUI		0	f	

Example 2

```
WITH foo AS (
  SELECT
    ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/
    loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
  *
FROM ST_BandMetadata(
  (SELECT rast FROM foo),
  ARRAY[1,3,2]::int[]
);
```

bandnum	pixeltype	nodatavalue	isoutdb	outdbbandnum	filesize	filetimestamp	path
1	8BUI		t				/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
3	8BUI		t				/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
2	8BUI		t				/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif



`ST_MetaData`, `ST_BandPixelType`

11.5.2 ST_BandNoDataValue

`ST_BandNoDataValue` — 返回指定栅格的 NODATA 值。如果栅格有多个波段，则返回指定波段的 NODATA 值。

Synopsis

double precision **ST_BandNoDataValue**(raster rast, integer bandnum=1);

返回

指定波段的 NODATA 值。

示例

```
SELECT ST_BandNoDataValue(rast,1) As bnval1,
       ST_BandNoDataValue(rast,2) As bnval2, ST_BandNoDataValue(rast,3) As bnval3
FROM dummy_rast
WHERE rid = 2;
```

bnval1	bnval2	bnval3
0	0	0

注意

ST_NumBands

11.5.3 ST_BandIsNoData

`ST_BandIsNoData` — 返回指定波段的 NODATA 值是否有效。

Synopsis

boolean **ST_BandIsNoData**(raster rast, integer band, boolean forceChecking=true);
boolean **ST_BandIsNoData**(raster rast, boolean forceChecking=true);

返回

如果指定波段的 NODATA 值有效，则返回 TRUE。如果指定波段的 NODATA 值无效，则返回 FALSE。如果指定波段的 NODATA 值无效，则返回 FALSE。如果指定波段的 NODATA 值无效，则返回 FALSE。

2.0.0 版本引入。

Note



如果指定波段的 NODATA 值有效，则返回 TRUE。如果指定波段的 NODATA 值无效，则返回 FALSE。如果指定波段的 NODATA 值无效，则返回 FALSE。如果指定波段的 NODATA 值无效，则返回 FALSE。

实验3.6.1

```
-- Create dummy table with one raster column
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ←
  = 3.
-- In the second band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'6' -- hasnodatavalue and isnodata value set to true.
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true
select st_bandisnodata(rast, 2) from dummy_rast where rid = 1; -- Expected false
```

实验3.6.1

ST_BandNoDataValue, ST_NumBands, ST_SetBandNoDataValue, ST_SetBandIsNoData

11.5.4 ST_BandPath

`ST_BandPath` — Returns the path to the band file. `bandnum` defaults to 1.

Synopsis

text **ST_BandPath**(raster rast, integer bandnum=1);

`ST_BandPath`

DB `ST_BandPath`.

`ST_BandPath`

`ST_BandPath`

11.5.5 ST_BandFileSize

`ST_BandFileSize` — Returns the file size of a band stored in file system. If no `bandnum` specified, 1 is assumed.

Synopsis

bigint **ST_BandFileSize**(raster rast, integer bandnum=1);

`ST_BandFileSize`

Returns the file size of a band stored in file system. Throws an error if called with an in db band, or if outdb access is not enabled.

This function is typically used in conjunction with `ST_BandPath()` and `ST_BandFileTimestamp()` so a client can determine if the filename of a outdb raster as seen by it is the same as the one seen by the server.

Availability: 2.5.0

`ST_BandFileSize`

```
SELECT ST_BandFileSize(rast,1) FROM dummy_rast WHERE rid = 1;
```

```
st_bandfilesize
-----
240574
```

11.5.6 ST_BandFileTimestamp

ST_BandFileTimestamp — Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.

Synopsis

bigint **ST_BandFileTimestamp**(raster rast, integer bandnum=1);

返回

Returns the file timestamp (number of seconds since Jan 1st 1970 00:00:00 UTC) of a band stored in file system. Throws an error if called with an in db band, or if outdb access is not enabled.

This function is typically used in conjunction with **ST_BandPath()** and **ST_BandFileSize()** so a client can determine if the filename of a outdb raster as seen by it is the same as the one seen by the server.

Availability: 2.5.0

示例

```
SELECT ST_BandFileTimestamp(rast,1) FROM dummy_rast WHERE rid = 1;
```

```
st_bandfiletimestamp
-----
1521807257
```

11.5.7 ST_BandPixelType

ST_BandPixelType — 返回指定栅格的像素数据类型。bandnum 指定要查询的波段，默认为 1。

Synopsis

text **ST_BandPixelType**(raster rast, integer bandnum=1);

返回

Returns name describing data type and size of values stored in each cell of given band.

11 返回的字符串格式如下：

- 1BB - 1 字节
- 2BUI - 2 字节的无符号整数
- 4BUI - 4 字节的无符号整数
- 8BSI - 8 字节的有符号整数
- 8BUI - 8 字节的无符号整数
- 16BSI - 16 字节的有符号整数
- 16BUI - 16 字节的无符号整数

- 32BSI - 32 有符号短整数
- 32BUI - 32 无符号短整数
- 32BF - 32 单精度浮点
- 64BF - 64 双精度浮点

例如

```
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

例如

ST_NumBands

11.5.8 ST_MinPossibleValue

ST_MinPossibleValue — 返回像素类型的可能的最小值。

Synopsis

integer **ST_MinPossibleValue**(text pixeltype);

例如

返回 16 位有符号短整数的可能的最小值。

例如

```
SELECT ST_MinPossibleValue('16BSI');
```

st_minpossiblevalue
-32768

```
SELECT ST_MinPossibleValue('8BUI');
```

st_minpossiblevalue
0

ST_BandPixelType

11.5.9 ST_HasNoBand

ST_HasNoBand — 检查栅格是否具有指定的波段。如果具有，则返回 1，否则返回 0。

Synopsis

boolean **ST_HasNoBand**(raster rast, integer bandnum=1);

检查栅格是否具有指定的波段。如果具有，则返回 1，否则返回 0。
2.0.0 版本引入。

```
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 栅格操作函数 (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。

例

```
-- get raster pixel polygon
SELECT i,j, ST_AsText(ST_PixelAsPolygon(foo.rast, i,j)) As blpgeom
FROM dummy_rast As foo
      CROSS JOIN generate_series(1,2) As i
      CROSS JOIN generate_series(1,1) As j
WHERE rid=2;
```

i	j	blpgeom
1	1	POLYGON((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...
2	1	POLYGON((3427927.8 5793244,3427927.85 5793244,3427927.85 5793243.95, ..

例

[ST_DumpAsPolygons](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#), [ST_Intersection](#), [ST_AsText](#)

11.6.2 ST_PixelAsPolygons

`ST_PixelAsPolygons` — 将栅格像素转换为多边形。返回 X, Y 坐标的多边形。

Synopsis

setof record **ST_PixelAsPolygons**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

例

返回记录格式: *geom geometry*, *val* double precision, *x* integer, *y* integers.



Note

When `exclude_nodata_value = TRUE`, only those pixels whose values are not NODATA are returned as points.



Note

`ST_PixelAsPolygons` 返回多边形。与 `ST_DumpAsPolygons` 返回点。

2.0.0 版本引入。

版本: 2.1.0 版本引入 `exclude_nodata_value` 参数。

版本: 2.1.1 版本引入 `exclude_nodata_value` 参数。

例

```
-- get raster pixel polygon
SELECT (gv).x, (gv).y, (gv).val, ST_AsText((gv).geom) geom
FROM (SELECT ST_PixelAsPolygons(
          ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.001, ←
          -0.001, 0.001, 0.001, 4269),
          '8BUI'::text, 1, 0),
          2, 2, 10),
          1, 1, NULL)
) gv
) foo;
```

x	y	val	geom
1	1		POLYGON((0 0,0.001 0.001,0.002 0,0.001 -0.001,0 0))
1	2	1	POLYGON((0.001 -0.001,0.002 0,0.003 -0.001,0.002 -0.002,0.001 -0.001))
2	1	1	POLYGON((0.001 0.001,0.002 0.002,0.003 0.001,0.002 0,0.001 0.001))
2	2	10	POLYGON((0.002 0,0.003 0.001,0.004 0,0.003 -0.001,0.002 0))

例

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#), [ST_AsText](#)

11.6.3 ST_PixelAsPoint

`ST_PixelAsPoint` — 从栅格中提取点几何。

Synopsis

geometry **ST_PixelAsPoint**(raster rast, integer columnx, integer rowy);

例

从栅格中提取点几何。

2.1.0 从栅格中提取点几何。

例

```
SELECT ST_AsText(ST_PixelAsPoint(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;
```

st_astext
POINT(0.5 0.5)

例

[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 坐标和像素值。如果指定了 exclude_nodata_value，则只返回非 NODATA 的像素。

Synopsis

setof record **ST_PixelAsPoints**(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.

2.1.0 版本引入。

2.1.1 版本引入 `exclude_nodata_value` 参数。

返回

```
SELECT x, y, val, ST_AsText(geom) FROM (SELECT (ST_PixelAsPoints(rast, 1)).* FROM dummy_rast WHERE rid = 2) foo;
```

x	y	val	st_astext
1	1	253	POINT(3427927.75 5793244)
2	1	254	POINT(3427927.8 5793244)
3	1	253	POINT(3427927.85 5793244)
4	1	254	POINT(3427927.9 5793244)
5	1	254	POINT(3427927.95 5793244)
1	2	253	POINT(3427927.75 5793243.95)
2	2	254	POINT(3427927.8 5793243.95)
3	2	254	POINT(3427927.85 5793243.95)
4	2	253	POINT(3427927.9 5793243.95)
5	2	249	POINT(3427927.95 5793243.95)
1	3	250	POINT(3427927.75 5793243.9)
2	3	254	POINT(3427927.8 5793243.9)
3	3	254	POINT(3427927.85 5793243.9)
4	3	252	POINT(3427927.9 5793243.9)
5	3	249	POINT(3427927.95 5793243.9)
1	4	251	POINT(3427927.75 5793243.85)
2	4	253	POINT(3427927.8 5793243.85)
3	4	254	POINT(3427927.85 5793243.85)
4	4	254	POINT(3427927.9 5793243.85)
5	4	253	POINT(3427927.95 5793243.85)
1	5	252	POINT(3427927.75 5793243.8)
2	5	250	POINT(3427927.8 5793243.8)
3	5	254	POINT(3427927.85 5793243.8)
4	5	254	POINT(3427927.9 5793243.8)
5	5	254	POINT(3427927.95 5793243.8)

函数

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#)

11.6.5 ST_PixelAsCentroid

`ST_PixelAsCentroid` — 返回栅格像素 (x, y) 的几何体。

Synopsis

geometry **ST_PixelAsCentroid**(raster rast, integer x, integer y);

函数

返回栅格像素 (x, y) 的几何体。

版本: 2.1.0 引入 C 语言实现。

2.1.0 版本引入。

函数

```
SELECT ST_AsText(ST_PixelAsCentroid(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;
```

```
st_astext
-----
POINT(1.5 2)
```

函数

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroids](#)

11.6.6 ST_PixelAsCentroids

`ST_PixelAsCentroids` — 返回栅格像素 (x, y) 的几何体 X, Y 坐标。

Synopsis

setof record **ST_PixelAsCentroids**(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.

- 2.1.0 添加 C 语言支持。
- 2.1.1 添加 `exclude_nodata_value` 参数。
- 2.1.0 添加 `geom` 参数。

返回

```
--LATERAL syntax requires PostgreSQL 9.3+
SELECT x, y, val, ST_AsText(geom)
FROM (SELECT dp.* FROM dummy_rast, LATERAL ST_PixelAsCentroids(rast, 1) AS dp WHERE rid <= 2) foo;
```

x	y	val	st_astext
1	1	253	POINT(3427927.775 5793243.975)
2	1	254	POINT(3427927.825 5793243.975)
3	1	253	POINT(3427927.875 5793243.975)
4	1	254	POINT(3427927.925 5793243.975)
5	1	254	POINT(3427927.975 5793243.975)
1	2	253	POINT(3427927.775 5793243.925)
2	2	254	POINT(3427927.825 5793243.925)
3	2	254	POINT(3427927.875 5793243.925)
4	2	253	POINT(3427927.925 5793243.925)
5	2	249	POINT(3427927.975 5793243.925)
1	3	250	POINT(3427927.775 5793243.875)
2	3	254	POINT(3427927.825 5793243.875)
3	3	254	POINT(3427927.875 5793243.875)
4	3	252	POINT(3427927.925 5793243.875)
5	3	249	POINT(3427927.975 5793243.875)
1	4	251	POINT(3427927.775 5793243.825)
2	4	253	POINT(3427927.825 5793243.825)
3	4	254	POINT(3427927.875 5793243.825)
4	4	254	POINT(3427927.925 5793243.825)
5	4	253	POINT(3427927.975 5793243.825)
1	5	252	POINT(3427927.775 5793243.775)
2	5	250	POINT(3427927.825 5793243.775)
3	5	254	POINT(3427927.875 5793243.775)
4	5	254	POINT(3427927.925 5793243.775)
5	5	254	POINT(3427927.975 5793243.775)

返回

`ST_DumpAsPolygons`, `ST_PixelAsPolygon`, `ST_PixelAsPolygons`, `ST_PixelAsPoint`, `ST_PixelAsPoints`, `ST_Pixel`

11.6.7 ST_Value

ST_Value — 返回 raster 在 columnx, rowy 处的值。如果 raster 是 32 位浮点型，则返回浮点型；如果是 32 位整数，则返回整数。exclude_nodata_value 参数用于控制是否返回 nodata 值。如果为 true，则返回 nodata 值；如果为 false，则返回 null。exclude_nodata_value 参数在 PostGIS 2.0.0 版本中引入。

Synopsis

```
double precision ST_Value(raster rast, geometry pt, boolean exclude_nodata_value=true);
double precision ST_Value(raster rast, integer band, geometry pt, boolean exclude_nodata_value=true,
text resample='nearest');
double precision ST_Value(raster rast, integer x, integer y, boolean exclude_nodata_value=true);
double precision ST_Value(raster rast, integer band, integer x, integer y, boolean exclude_nodata_value=true);
```

参数

columnx, rowy 是栅格的列索引和行索引，从 1 开始。exclude_nodata_value 是布尔值，用于控制是否返回 nodata 值。如果为 true，则返回 nodata 值；如果为 false，则返回 null。resample 是文本参数，用于指定重采样方法，可以是 'nearest' 或 'bilinear'。

The allowed values of the resample parameter are "nearest" which performs the default nearest-neighbor resampling, and "bilinear" which performs a **bilinear interpolation** to estimate the value between pixel centers.

PostGIS 2.1.0 版本引入 exclude_nodata_value 参数。

PostGIS 2.0.0 版本引入 exclude_nodata_value 参数。

示例

```
-- get raster values at particular postgis geometry points
-- the srid of your geometry should be same as for your raster
SELECT rid, ST_Value(rast, foo.pt_geom) As b1pval, ST_Value(rast, 2, foo.pt_geom) As b2pval
FROM dummy_rast CROSS JOIN (SELECT ST_SetSRID(ST_Point(3427927.77, 5793243.76), 0) As
pt_geom) As foo
WHERE rid=2;
```

rid	b1pval	b2pval
2	252	79

```
-- general fictitious example using a real table
SELECT rid, ST_Value(rast, 3, sometable.geom) As b3pval
FROM sometable
WHERE ST_Intersects(rast,sometable.geom);
```

```
SELECT rid, ST_Value(rast, 1, 1, 1) As b1pval,
ST_Value(rast, 2, 1, 1) As b2pval, ST_Value(rast, 3, 1, 1) As b3pval
FROM dummy_rast
WHERE rid=2;
```

rid	b1pval	b2pval	b3pval
2	253	78	70

```

--- Get all values in bands 1,2,3 of each pixel --
SELECT x, y, ST_Value(rast, 1, x, y) As b1val,
       ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1, 1000) As x CROSS JOIN generate_series(1, 1000) As y
WHERE rid = 2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);

```

x	y	b1val	b2val	b3val
1	1	253	78	70
1	2	253	96	80
1	3	250	99	90
1	4	251	89	77
1	5	252	79	62
2	1	254	98	86
2	2	254	118	108
:				
:				

```

--- Get all values in bands 1,2,3 of each pixel same as above but returning the upper left ←
point point of each pixel --
SELECT ST_AsText(ST_SetSRID(
  ST_Point(ST_UpperLeftX(rast) + ST_ScaleX(rast)*x,
    ST_UpperLeftY(rast) + ST_ScaleY(rast)*y),
    ST_SRID(rast))) As uplpt
, ST_Value(rast, 1, x, y) As b1val,
  ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid = 2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);

```

uplpt	b1val	b2val	b3val
POINT(3427929.25 5793245.5)	253	78	70
POINT(3427929.25 5793247)	253	96	80
POINT(3427929.25 5793248.5)	250	99	90
:			

```

--- Get a polygon formed by union of all pixels
that fall in a particular value range and intersect particular polygon --
SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
  ST_UpperLeftX(rast), ST_UpperLeftY(rast),
  ST_UpperLeftX(rast) + ST_ScaleX(rast),
  ST_UpperLeftY(rast) + ST_ScaleY(rast), 0
), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid = 2
AND x <= ST_Width(rast) AND y <= ST_Height(rast)) As foo
WHERE
  ST_Intersects(
    pixpolyg,
    ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928 ←
5793243.75,3427928 5793244))',0)

```

```
) AND b2val != 254;
```

```
shadow
```

```
-----
MULTIPOLYGON(((3427928 5793243.9,3427928 5793243.85,3427927.95 5793243.85,3427927.95  ←
5793243.9,
3427927.95 5793243.95,3427928 5793243.95,3427928.05 5793243.95,3427928.05  ←
5793243.9,3427928 5793243.9)),((3427927.95 5793243.9,3427927.95 579324
3.85,3427927.9 5793243.85,3427927.85 5793243.85,3427927.85 5793243.9,3427927.9  ←
5793243.9,3427927.9 5793243.95,
3427927.95 5793243.95,3427927.95 5793243.9)),((3427927.85 5793243.75,3427927.85  ←
5793243.7,3427927.8 5793243.7,3427927.8 5793243.75
,3427927.8 5793243.8,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85  ←
5793243.8,3427927.85 5793243.75))),
((3427928.05 5793243.75,3427928.05 5793243.7,3427928 5793243.7,3427927.95  ←
5793243.7,3427927.95 5793243.75,3427927.95 5793243.8,3427
927.95 5793243.85,3427928 5793243.85,3427928 5793243.8,3427928.05 5793243.8,
3427928.05 5793243.75)),((3427927.95 5793243.75,3427927.95 5793243.7,3427927.9  ←
5793243.7,3427927.85 5793243.7,
3427927.85 5793243.75,3427927.85 5793243.8,3427927.85 5793243.85,3427927.9 5793243.85,
3427927.95 5793243.85,3427927.95 5793243.8,3427927.95 5793243.75)))
```

```
--- Checking all the pixels of a large raster tile can take a long time.
--- You can dramatically improve speed at some lose of precision by orders of magnitude
-- by sampling pixels using the step optional parameter of generate_series.
-- This next example does the same as previous but by checking 1 for every 4 (2x2) pixels  ←
and putting in the last checked
-- putting in the checked pixel as the value for subsequent 4
```

```
SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
    ST_UpperLeftX(rast), ST_UpperLeftY(rast),
    ST_UpperLeftX(rast) + ST_ScaleX(rast)*2,
    ST_UpperLeftY(rast) + ST_ScaleY(rast)*2, 0
    ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
    ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
FROM dummy_rast CROSS JOIN
generate_series(1,1000,2) As x CROSS JOIN generate_series(1,1000,2) As y
WHERE rid = 2
AND x <= ST_Width(rast) AND y <= ST_Height(rast) ) As foo
WHERE
    ST_Intersects(
        pixpolyg,
        ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928  ←
5793243.75,3427928 5793244))',0)
    ) AND b2val != 254;
```

```
shadow
```

```
-----
MULTIPOLYGON(((3427927.9 5793243.85,3427927.8 5793243.85,3427927.8 5793243.95,
3427927.9 5793243.95,3427928 5793243.95,3427928.1 5793243.95,3427928.1 5793243.85,3427928  ←
5793243.85,3427927.9 5793243.85)),
((3427927.9 5793243.65,3427927.8 5793243.65,3427927.8 5793243.75,3427927.8  ←
5793243.85,3427927.9 5793243.85,
3427928 5793243.85,3427928 5793243.75,3427928.1 5793243.75,3427928.1 5793243.65,3427928  ←
5793243.65,3427927.9 5793243.65)))
```

函数

[ST_DumpValues](#), [ST_SetValue](#), [ST_DumpAsPolygons](#), [ST_NumBands](#), [ST_PixelAsPolygon](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_UpperLeftX](#), [ST_UpperLeftY](#), [ST_SRID](#), [ST_AsText](#), [ST_Point](#), [ST_MakeEnvelope](#), [ST_Intersects](#), [ST_Intersection](#)

11.6.8 ST_NearestValue

ST_NearestValue — columnx 行号, 返回指定栅格中最近的栅格单元值。如果指定了 columnx 和 rowy, 则返回指定单元格的值。如果指定了 columnx 和 rowy, 则返回指定单元格的值。如果指定了 columnx 和 rowy, 则返回指定单元格的值。

Synopsis

double precision **ST_NearestValue**(raster rast, integer bandnum, geometry pt, boolean exclude_nodata_value= false);
 double precision **ST_NearestValue**(raster rast, geometry pt, boolean exclude_nodata_value= true);
 double precision **ST_NearestValue**(raster rast, integer bandnum, integer columnx, integer rowy, boolean exclude_nodata_value= true);
 double precision **ST_NearestValue**(raster rast, integer columnx, integer rowy, boolean exclude_nodata_value= true);

函数

columnx, rowy 指定栅格单元, 返回指定栅格单元的值。如果指定了 columnx 和 rowy, 则返回指定单元格的值。如果指定了 columnx 和 rowy, 则返回指定单元格的值。如果指定了 columnx 和 rowy, 则返回指定单元格的值。

bandnum 指定栅格带, bandnum 1 指定第一带。exclude_nodata_value 指定是否排除 NODATA 值。nodata 指定是否返回 NODATA 值。exclude_nodata_value 指定是否排除 NODATA 值。

2.1.0 版本引入。



Note

ST_NearestValue 返回 ST_Value 返回的值。

函数

```
-- pixel 2x2 has value
SELECT
  ST_Value(rast, 2, 2) AS value,
  ST_NearestValue(rast, 2, 2) AS nearestvalue
FROM (
  SELECT
    ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_SetValue(
            ST_SetValue(
              ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
              ),
              1, 1, 0.
            )
          )
        )
      )
    )
```

```
        ),
        2, 3, 0.
    ),
    3, 5, 0.
),
4, 2, 0.
),
5, 4, 0.
) AS rast
) AS foo

value | nearestvalue
-----+-----
1 | 1
```

```
-- pixel 2x3 is NODATA
SELECT
    ST_Value(rast, 2, 3) AS value,
    ST_NearestValue(rast, 2, 3) AS nearestvalue
FROM (
    SELECT
        ST_SetValue(
            ST_SetValue(
                ST_SetValue(
                    ST_SetValue(
                        ST_SetValue(
                            ST_AddBand(
                                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                                '8BUI'::text, 1, 0
                            ),
                            1, 1, 0.
                        ),
                        2, 3, 0.
                    ),
                    3, 5, 0.
                ),
                4, 2, 0.
            ),
            5, 4, 0.
        ) AS rast
    ) AS foo

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);

国国

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

国国

```
--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
WITH test_raster AS (
SELECT
ST_SetValues(
  ST_AddBand(
    ST_MakeEmptyRaster(width => 2, height => 2,
      upperleftx => 0, upperlefty => 2,
      scalex => 1.0, scaley => -1.0,
      skewx => 0, skewy => 0, srid => 4326),
    index => 1, pixeltype => 'l6BSI',
    initialvalue => 0,
    nodataval => -999),
    1,1,1,
    newvalueset =>ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8] ←
      ]) AS rast
)
SELECT
ST_AsText(
  ST_SetZ(
    rast,
    band => 1,
    geom => 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)::geometry',
    resample => 'bilinear'
  ))
FROM test_raster

          st_astext
-----
LINESTRING Z (1 1.9 38,1 0.2 27)
```

国国

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);

国国

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

国国

```
--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
WITH test_raster AS (
SELECT
ST_SetValues(
  ST_AddBand(
    ST_MakeEmptyRaster(width => 2, height => 2,
      upperleftx => 0, upperlefty => 2,
      scalex => 1.0, scaley => -1.0,
      skewx => 0, skewy => 0, srid => 4326),
    index => 1, pixeltype => 'l6BSI',
    initialvalue => 0,
    nodataval => -999),
  1,1,1,
  newvalueset =>ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8] ←
    ]) AS rast
)
SELECT
ST_AsText(
  ST_SetM(
    rast,
    band => 1,
    geom => 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)::geometry',
    resample => 'bilinear'
  ))
FROM test_raster

          st_astext
-----
LINESTRING M (1 1.9 38,1 0.2 27)
```



```

        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast
    ) AS foo

    st_neighborhood
-----
{{NULL,1,1},{1,1,1},{1,NULL,1}}

```

```

-- pixel 2x3 is NODATA
SELECT
    ST_Neighborhood(rast, 2, 3, 1, 1)
FROM (
    SELECT
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast
    ) AS foo

    st_neighborhood
-----
{{1,1,1},{1,NULL,1},{1,1,1}}

```

```

-- pixel 3x3 has value
-- exclude_nodata_value = FALSE
SELECT
    ST_Neighborhood(rast, 3, 3, 1, 1, false)
FROM ST_SetValues(
    ST_AddBand(
        ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
        '8BUI'::text, 1, 0
    ),
    1, 1, 1, ARRAY[
        [0, 1, 1, 1, 1],
        [1, 1, 1, 0, 1],
        [1, 0, 1, 1, 1],
        [1, 1, 1, 1, 0],
        [1, 1, 0, 1, 1]
    ]::double precision[],

```

```
        1
    ) AS rast

    st_neighborhood
-----
{{1,1,0},{0,1,1},{1,1,1}}
```

函数

[ST_NearestValue](#), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Mean4ma](#), [ST_Range4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

11.6.12 ST_SetValue

ST_SetValue — 将列中的指定像素值设置为新值。如果未指定列，则使用默认列。如果未指定行，则使用默认行。如果未指定几何，则使用默认几何。

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);

函数

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.

更新: 2.1.0 添加 ST_SetValue() 函数。添加 ST_SetValues() 函数 (wrapper)。

函数

```
-- Geometry example
SELECT (foo.geomval).val, ST_AsText(ST_Union((foo.geomval).geom))
FROM (SELECT ST_DumpAsPolygons(
        ST_SetValue(rast,1,
                    ST_Point(3427927.75, 5793243.95),
                    50)
    ) As geomval
FROM dummy_rast
where rid = 2) As foo
WHERE (foo.geomval).val < 250
GROUP BY (foo.geomval).val;
```

val	st_astext
50	POLYGON((3427927.75 5793244,3427927.75 5793243.95,3427927.8 579324 ...
249	POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 57932 ...

```
-- Store the changed raster --
UPDATE dummy_rast SET rast = ST_SetValue(rast,1, ST_Point(3427927.75, 5793243.95),100)
WHERE rid = 2 ;
```



ST Value, ST DumpAsPolygons

11.6.13 ST_SetValues

ST SetValues —

Synopsis

```

raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[][])
newvalueset, boolean[][]) noset=NULL, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[][])
newvalueset, double precision nosetvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, integer width, integer
height, double precision newvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer columnx, integer rowy, integer width, integer height, double
precision newvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, geomval[] geomvalset, boolean keepnodata=FALSE);

```



XXXXXXXXXX, XXXXXXXXXXXXXXX (X) XXXXXXXXXXXXXXXXXXXXXXXXXXXX.

```
keepnodata TRUE ###, NODATA ##### newvalueset #####
```

1 1, columnx, rowy newvalueset newvalueset newvalueset noset (PostgreSQL 1.1.1).

```

2 1 1, noset nosetvalue . newvalueset nosetvalue . 2 .

```

3 columns, columnx, rowy columns, width height columns. 3 columns

4 rast 3

5 行, 列の行列を **geomval** で指定する。 POINT 列 MULTIPOINT 列, 列の型は指定しない。 列の型は指定しない。 5 行の行列。

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.

图例: 图 1

```
/*
The ST_SetValues() does the following...

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 1 | 1 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      =
> | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +

*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
    SELECT
        ST_PixelAsPolygons(
            ST_SetValues(
                ST_AddBand(
                    ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 2, 2, ARRAY[[9, 9], [9, 9]]::double precision[][]
            )
        ) AS poly
    ) foo
ORDER BY 1, 2;

x | y | val
---+---+---
1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 | 9
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9
```

```
/*
The ST_SetValues() does the following...

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      =
> | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +

*/
SELECT
    (poly).x,
```

```
(poly).y,
(poly).val
FROM (
SELECT
  ST_PixelAsPolygons(
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
        1, '8BUI', 1, 0
      ),
      1, 1, 1, ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][]
    )
  ) AS poly
) foo
ORDER BY 1, 2;
```

x	y	val
1	1	9
1	2	9
1	3	9
2	1	9
2	2	
2	3	9
3	1	9
3	2	9
3	3	9

```
/*
The ST_SetValues() does the following...

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      => | 1 |   | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +

*/
SELECT
  (poly).x,
  (poly).y,
  (poly).val
FROM (
SELECT
  ST_PixelAsPolygons(
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
        1, '8BUI', 1, 0
      ),
      1, 1, 1,
      ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[[[]],
      ARRAY[[false], [true]]::boolean[[[]]
    )
  ) AS poly
) foo
ORDER BY 1, 2;

x | y | val
---+---+---
```

1	1	9
1	2	1
1	3	9
2	1	9
2	2	
2	3	9
3	1	9
3	2	9
3	3	9

```
/*
The ST_SetValues() does the following...
```

+ - + - + - +		+ - + - + - +
1 1		9 9
+ - + - + - +		+ - + - + - +
1 1 1	=>	1 9
+ - + - + - +		+ - + - + - +
1 1 1		9 9 9
+ - + - + - +		+ - + - + - +

```
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
    SELECT
        ST_PixelAsPolygons(
            ST_SetValues(
                ST_SetValue(
                    ST_AddBand(
                        ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                        1, '8BUI', 1, 0
                    ),
                    1, 1, 1, NULL
                ),
                1, 1, 1,
                ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
                ARRAY[[false], [true]]::boolean[][],
                TRUE
            )
        ) AS poly
    ) foo
ORDER BY 1, 2;
```

x	y	val
1	1	
1	2	1
1	3	9
2	1	9
2	2	
2	3	9
3	1	9
3	2	9
3	3	9

```

/*
The ST_SetValues() does the following...

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 1 | 1 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      => | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
    SELECT
        ST_PixelAsPolygons(
            ST_SetValues(
                ST_AddBand(
                    ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 1, 1, ARRAY[[-1, -1, -1], [-1, 9, 9], [-1, 9, 9]]::double precision[3][3], -1
            )
        ) AS poly
    ) foo
ORDER BY 1, 2;

x | y | val
---+---+---
1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 | 9
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9

```

```

/*
This example is like the previous one. Instead of nosetvalue = -1, nosetvalue = NULL

The ST_SetValues() does the following...

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 1 | 1 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      => | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
    SELECT
        ST_PixelAsPolygons(

```



```
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[NULL, NULL, NULL], [NULL, 9, 9], [NULL, 9, 9]]::double precision ←
                precision[][], NULL::double precision
        )
    ) AS poly
) foo
ORDER BY 1, 2;
```

x	y	val
1	1	1
1	2	1
1	3	1
2	1	1
2	2	9
2	3	9
3	1	1
3	2	9
3	3	9

图 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

练习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
  t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2.gid), ↵
    ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 2
      AND t3.gid = 1
ORDER BY t1.rid, t2.gid, t3.gid;
```

rid	gid	gid	st_dumpvalues
1	2	1	(1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,1,2,NULL},{ ↵ NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")

(1 row)

[ST_Value](#), [ST_SetValue](#), [ST_PixelAsPolygons](#)

11.6.14 ST_DumpValues

ST_DumpValues — 将 2 维栅格值拆分为行。

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);

将 2 维栅格值拆分为行 (行, 列)。 nband 为 NULL 时
返回所有波段, 否则返回指定波段。

2.1.0 引入。

```
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}}

```
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}}

```
WITH foo AS (
    SELECT ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI ←
        ', 1, 0), 1, 2, 5) AS rast
)
SELECT
    (ST_DumpValues(rast, 1))[2][1]
FROM foo;
```

```
st_dumpvalues
-----
                    5
(1 row)
```


ST Value, ST SetValue, ST SetValues

11.6.15 ST_PixelOfValue

ST PixelOfValue — columnx, rowy .

Synopsis

```
setof record ST_PixelOfValue( raster rast , integer nband , double precision[] search , boolean 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
255	5	4
1	5	5

11.7 地理参考

11.7.1 ST_SetGeoReference

`ST_SetGeoReference` — 将地理参考信息设置到 6 个参数。支持 GDAL 和 ESRI 两种格式。GDAL 格式为：

Synopsis

`raster` **ST_SetGeoReference**(`raster rast`, `text georefcoords`, `text format=GDAL`);
`raster` **ST_SetGeoReference**(`raster rast`, `double precision upperleftx`, `double precision upperlefty`,
`double precision scalex`, `double precision scaley`, `double precision skewx`, `double precision skewy`);

格式：

支持 GDAL 和 ESRI 两种格式。'GDAL' 和 'ESRI' 格式分别对应 GDAL 和 ESRI 格式。6 个参数分别对应 NULL 值。

支持的格式：

GDAL:

```
scalex skewy skewx scaley upperleftx upperlefty
```

ESRI:

```
scalex skewy skewx scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5
```



Note

在 PostgreSQL 12 之前，此函数在地理数据库（如 PostgreSQL）中不可用。从 PostgreSQL 12 开始，此函数在地理数据库（如 PostgreSQL）中可用。

从 2.1.0 版本开始，`ST_SetGeoReference(raster, double precision, ...)` 函数可用。

示例：

```
WITH foo AS (
  SELECT ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0) AS rast
)
SELECT
  0 AS rid, (ST_Metadatas(rast)).*
FROM foo
UNION ALL
SELECT
  1, (ST_Metadatas(ST_SetGeoReference(rast, '10 0 0 -10 0.1 0.1', 'GDAL'))).*
FROM foo
UNION ALL
SELECT
  2, (ST_Metadatas(ST_SetGeoReference(rast, '10 0 0 -10 5.1 -4.9', 'ESRI'))).*
FROM foo
UNION ALL
SELECT
```

```
3, (ST_Metadata(ST_SetGeoReference(rast, 1, 1, 10, -10, 0.001, 0.001))).*
FROM foo
```

rid	upperleftx skewy	srid	numbands	upperlefty	width	height	scalex	scaley	skewx	
0	0	0	0	0	5	5	1	-1	0	↔
1	0	0	0.1	0.1	5	5	10	-10	0	↔
2	0.09999999999999996	0.09999999999999996			5	5	10	-10	0	↔
3	0.001	0	1	1	5	5	10	-10	0.001	↔

↔

[ST_GeoReference](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_UpperLeftX](#), [ST_UpperLeftY](#)

11.7.2 ST_SetRotation

ST_SetRotation — 将栅格旋转到指定的角度。

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，则输出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;
```

st_scalex	st_scaley	st_skewx	st_skewy	
st_scalex	st_scaley	st_skewx	st_skewy	
-1.51937582571764	-2.27906373857646	1.95086352047135	1.30057568031423	↔
2	3	0	0	
-0.0379843956429411	-0.0379843956429411	0.0325143920078558	0.0325143920078558	↔
0.05	-0.05	0	0	

↔

[ST_Rotation](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SkewX](#), [ST_SkewY](#)

11.7.3 ST_SetScale

ST_SetScale — X 和 Y 坐标乘以指定的标度因子。返回具有相同几何类型的几何体。

Synopsis

```
raster ST_SetScale(raster rast, float8 xy);
raster ST_SetScale(raster rast, float8 x, float8 y);
```

返回

X 和 Y 坐标乘以指定的标度因子。返回具有相同几何类型的几何体。如果标度因子为负数，则几何体将被镜像。ST_SetScale 与 ST_Rescale 类似，但 ST_SetScale 只接受一个标度因子。



Note

ST_SetScale 与 ST_Rescale 类似，但 ST_SetScale 只接受一个标度因子。ST_SetScale 与 ST_Rescale 类似，但 ST_SetScale 只接受一个标度因子。ST_SetScale 与 ST_Rescale 类似，但 ST_SetScale 只接受一个标度因子。

从 2.0.0 开始，WKTRaster 类型的 ST_SetPixelSize 函数已弃用。2.0.0 版本中，ST_SetPixelSize 函数已弃用。

示例

```
UPDATE dummy_rast
  SET rast = ST_SetScale(rast, 1.5)
WHERE rid = 2;

SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
WHERE rid = 2;
```

pixx	pixy	newbox
1.5	1.5	BOX(3427927.75 5793244 0, 3427935.25 5793251.5 0)

```
UPDATE dummy_rast
  SET rast = ST_SetScale(rast, 1.5, 0.55)
WHERE rid = 2;

SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
WHERE rid = 2;
```

pixx	pixy	newbox
1.5	0.55	BOX(3427927.75 5793244 0, 3427935.25 5793247 0)

返回

ST_ScaleX, ST_ScaleY, Box3D

11.7.4 ST_SetSkew

`ST_SetSkew` — 设置 X 和 Y 的偏斜 (skew) 值。 设置 X 和 Y 的偏斜值。

Synopsis

raster **ST_SetSkew**(raster rast, float8 skewxy);
raster **ST_SetSkew**(raster rast, float8 skewx, float8 skewy);

返回

设置 X 和 Y 的偏斜值 (skew) 值。 设置 X 和 Y 的偏斜值。 设置 X 和 Y 的偏斜值。

返回

```
-- Example 1
UPDATE dummy_rast SET rast = ST_SetSkew(rast,1,2) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

rid	skewx	skewy	georef
1	1	2	2.0000000000 : 2.0000000000 : 1.0000000000 : 3.0000000000 : 0.5000000000 : 0.5000000000

```
-- Example 2 set both to same number:
UPDATE dummy_rast SET rast = ST_SetSkew(rast,0) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

rid	skewx	skewy	georef
1	0	0	2.0000000000 : 0.0000000000 : 0.0000000000 : 3.0000000000 : 0.5000000000 : 0.5000000000

返回

[ST_GeoReference](#), [ST_SetGeoReference](#), [ST_SkewX](#), [ST_SkewY](#)

11.7.5 ST_SetSRID

ST_SetSRID — 设置 spatial_ref_sys 表中的 SRID 值。

Synopsis

```
raster ST_SetSRID(raster rast, integer srid);
```

返回

具有指定 SRID 的 raster。



Note

如果 raster 已经具有 SRID，则 ST_SetSRID 将不会更改 SRID 值。如果 raster 没有 SRID，则 ST_SetSRID 将使用指定的 SRID 值。

参见

Section 4.5, [ST_SRID](#)

11.7.6 ST_SetUpperLeft

ST_SetUpperLeft — 设置栅格的左上角像素的投影 X 和 Y 坐标。

Synopsis

```
raster ST_SetUpperLeft(raster rast, double precision x, double precision y);
```

返回

具有指定 X 和 Y 坐标的栅格。

参见

```
SELECT ST_SetUpperLeft(rast, -71.01, 42.37)
FROM dummy_rast
WHERE rid = 2;
```

参见

[ST_UpperLeftX](#), [ST_UpperLeftY](#)

11.7.7 ST_Resample

ST_Resample — 对栅格进行重采样，指定新的宽度和高度，以及可选的网格大小、缩放、倾斜和最大误差。返回具有指定参数的重采样栅格。

Synopsis

```
raster ST_Resample(raster rast, integer width, integer height, double precision gridx=NULL, double
precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor,
double precision maxerr=0.125);
raster ST_Resample(raster rast, double precision scalex=0, double precision scaley=0, double preci-
sion gridx=NULL, double precision gridy=NULL, double precision skewx=0, double precision skewy=0,
text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_Resample(raster rast, raster ref, text algorithm=NearestNeighbor, double precision max-
err=0.125, boolean usescale=true);
raster ST_Resample(raster rast, raster ref, boolean usescale, text algorithm=NearestNeighbor, dou-
ble precision maxerr=0.125);
```

参数

`width` 和 `height` (宽度和高度), `gridx` 和 `gridy` (网格大小), `scalex`, `scaley`, `skewx` 和 `skewy` (缩放、倾斜和最大误差) 是可选的。如果提供了 `ref`，则 `usescale` 参数将使用 `ref` 的 SRID 值。

New pixel values are computed using one of the following resampling algorithms:

- NearestNeighbor (english or american spelling)
- Bilinear
- Cubic
- CubicSpline
- Lanczos
- Max
- Min

The default is NearestNeighbor which is the fastest but results in the worst interpolation.

`maxerr` 参数默认为 0.125。



Note

有关 GDAL Warp 重采样方法的更多信息，请参阅 [GDAL Warp resampling methods](#)。

2.0.0 版本引入。GDAL 1.6.1 版本引入。

Enhanced: 3.4.0 max and min resampling options added

图 11.7.7

```
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

图 11.7.8

[ST_Rescale](#), [ST_Resize](#), [ST_Transform](#)

11.7.8 ST_Rescale

ST_Rescale — Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline, Lanczos, Max or Min resampling algorithm. Default is NearestNeighbor.

Synopsis

```
raster ST_Rescale(raster rast, double precision scalex, text algorithm=NearestNeighbor, double
precision maxerr=0.125);
raster ST_Rescale(raster rast, double precision scalex, double precision scaley, text algorithm=NearestNeighbor,
double precision maxerr=0.125);
```

图 11.7.9

Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using one of the following resampling algorithms:

- NearestNeighbor (english or american spelling)
- Bilinear
- Cubic
- CubicSpline
- Lanczos
- Max

- Min

The default is NearestNeighbor which is the fastest but results in the worst interpolation.

scalex and scaley define the new pixel size. scaley must often be negative to get well oriented raster.

scalex 和 scaley 指定新的像素大小，scaley 经常为负数以获得正确的方向。ST_Resize 使用最近邻插值。

maxerr 是变换近似的阈值（以像素为单位）。如果未指定 maxerr，则使用默认值 0.125，这是 GDAL gdalwarp 工具使用的值。如果设置为零，则不进行近似。



Note

GDAL Warp resampling methods 参考。



Note

ST_Rescale 使用 ST_SetScale 设置新的像素大小。ST_SetScale 设置新的像素大小（scalex 和 scaley）。ST_Rescale 使用 ST_SetScale 设置新的像素大小。ST_SetScale 设置新的像素大小。

2.0.0 版本。GDAL 1.6.1 版本。

Enhanced: 3.4.0 max and min resampling options added

更新: 2.1.0 版本 SRID 参考。

返回

返回 0.001 和 0.0015 的像素大小。

```
-- the original raster pixel size
SELECT ST_PixelWidth(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269), '8BUI'::text, 1, 0)) width

width
-----
0.001

-- the rescaled raster raster pixel size
SELECT ST_PixelWidth(ST_Rescale(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0015)) width

width
-----
0.0015
```

返回

ST_Resize, ST_Resample, ST_SetScale, ST_ScaleX, ST_ScaleY, ST_Transform

11.7.9 ST_Reskew

ST_Reskew — 对栅格 (栅格数据集) 进行重新采样。 NearestNeighbor(最近邻), Bilinear, Cubic, CubicSpline 和 Lanczos 是重新采样的方法。 NearestNeighbor 是默认方法。

Synopsis

```
raster ST_Reskew(raster rast, double precision skewxy, text algorithm=NearestNeighbor, double
precision maxerr=0.125);
raster ST_Reskew(raster rast, double precision skewx, double precision skewy, text algorithm=NearestNeighbor,
double precision maxerr=0.125);
```

参数

skewx 和 **skewy** 是重新采样的角度。 NearestNeighbor(最近邻), Bilinear, Cubic, CubicSpline 和 Lanczos 是重新采样的方法。 NearestNeighbor 是默认方法。

skewx 和 **skewy** 是重新采样的角度。

重新采样的角度。

maxerr 是重新采样的最大误差 0.125 是默认值。



Note

重新采样的 [GDAL Warp resampling methods](#) 是最近邻。



Note

ST_Reskew 函数在 3.6.0 版本中引入。 **ST_SetSkew** 函数在 3.6.0 版本中引入。 **ST_SetSkew** 函数在 3.6.0 版本中引入。 **ST_Reskew** 函数在 3.6.0 版本中引入。 **ST_SetSkew** 函数在 3.6.0 版本中引入。

2.0.0 版本中引入。 GDAL 1.6.1 版本中引入。

版本: 2.1.0 版本中引入 SRID 函数中引入。

参数

角度 0.0 和 0.0015 是重新采样的角度。

```
-- the original raster non-rotated
SELECT ST_Rotation(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));

-- result
0

-- the reskewed raster raster rotation
SELECT ST_Rotation(ST_Reskew(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, ←
0, 0, 4269), '8BUI'::text, 1, 0), 0.0015));

-- result
-0.982793723247329
```

函数

[ST_Resample](#), [ST_Rescale](#), [ST_SetSkew](#), [ST_SetRotation](#), [ST_SkewX](#), [ST_SkewY](#), [ST_Transform](#)

11.7.10 ST_SnapToGrid

`ST_SnapToGrid` — 将栅格数据对齐到指定的网格。 `NearestNeighbor`(`栅格`, `算法`), `Bilinear`, `Cubic`, `CubicSpline` 或 `Lanczos` 重采样方法。 `NearestNeighbor` 是默认值。

Synopsis

```
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex=DEFAULT 0, double precision scaley=DEFAULT 0);
```

```
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

```
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

参数

`gridx` & `gridy` (栅格) 的 `scalex` & `scaley` 的 `NearestNeighbor`(`算法`), `Bilinear`, `Cubic`, `CubicSpline` 或 `Lanczos` 重采样方法。 `NearestNeighbor` 是默认值。

`gridx` & `gridy` 栅格的 `scalex` & `scaley` 的 `NearestNeighbor`(`算法`), `Bilinear`, `Cubic`, `CubicSpline` 或 `Lanczos` 重采样方法。 `NearestNeighbor` 是默认值。

You can optionally define the pixel size of the new grid with `scalex` and `scaley`.

`scalex` & `scaley` 栅格的 `scalex` & `scaley` 的 `NearestNeighbor`(`算法`), `Bilinear`, `Cubic`, `CubicSpline` 或 `Lanczos` 重采样方法。 `NearestNeighbor` 是默认值。

`maxerr` 栅格的 `scalex` & `scaley` 的 `NearestNeighbor`(`算法`), `Bilinear`, `Cubic`, `CubicSpline` 或 `Lanczos` 重采样方法。 `NearestNeighbor` 是默认值。



Note

栅格的 `scalex` & `scaley` 的 `NearestNeighbor`(`算法`), `Bilinear`, `Cubic`, `CubicSpline` 或 `Lanczos` 重采样方法。 `NearestNeighbor` 是默认值。



Note

栅格的 `scalex` & `scaley` 的 `NearestNeighbor`(`算法`), `Bilinear`, `Cubic`, `CubicSpline` 或 `Lanczos` 重采样方法。 `NearestNeighbor` 是默认值。

2.0.0 栅格的 `scalex` & `scaley` 的 `NearestNeighbor`(`算法`), `Bilinear`, `Cubic`, `CubicSpline` 或 `Lanczos` 重采样方法。 `NearestNeighbor` 是默认值。

栅格: 2.1.0 栅格的 `scalex` & `scaley` 的 `NearestNeighbor`(`算法`), `Bilinear`, `Cubic`, `CubicSpline` 或 `Lanczos` 重采样方法。 `NearestNeighbor` 是默认值。

图 11.7.11

图 11.7.11 展示了 ST_UpperLeftX 和 ST_UpperLeftY 函数的使用。

```
-- the original raster upper left X
SELECT ST_UpperLeftX(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));
-- result
0

-- the upper left of raster after snapping
SELECT ST_UpperLeftX(ST_SnapToGrid(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, ←
-0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0002, 0.0002));

--result
-0.0008
```

图 11.7.11

[ST_Resample](#), [ST_Rescale](#), [ST_UpperLeftX](#), [ST_UpperLeftY](#)

11.7.11 ST_Resize

ST_Resize — 将栅格重新调整为指定的宽度和高度。

Synopsis

raster **ST_Resize**(raster rast, integer width, integer height, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resize**(raster rast, double precision percentwidth, double precision percentheight, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resize**(raster rast, text width, text height, text algorithm=NearestNeighbor, double precision maxerr=0.125);

图 11.7.11

图 11.7.11 展示了 ST_Resize 函数的使用。该函数将输入栅格的宽度和高度调整为指定的宽度和高度。如果指定的宽度和高度是百分比，则函数将根据输入栅格的原始宽度和高度进行计算。

NearestNeighbor(最近邻), Bilinear, Cubic, CubicSpline 和 Lanczos 是支持的插值算法。NearestNeighbor 是默认算法。

图 1 显示了最近邻插值的示例。

图 2 显示了双线性插值的示例。图 3 显示了三次插值的示例。

图 3 显示了三次插值的示例。图 4 显示了三次样条插值的示例。图 5 显示了 Lanczos 插值的示例。图 6 显示了最近邻插值的示例。

2.1.0 版本引入了 ST_Resize 函数。GDAL 1.6.1 版本引入了 ST_Resize 函数。

例

```
WITH foo AS(
SELECT
  1 AS rid,
  ST_Resize(
    ST_AddBand(
      ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
      , 1, '8BUI', 255, 0
    )
    , '50%', '500') AS rast
UNION ALL
SELECT
  2 AS rid,
  ST_Resize(
    ST_AddBand(
      ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
      , 1, '8BUI', 255, 0
    )
    , 500, 100) AS rast
UNION ALL
SELECT
  3 AS rid,
  ST_Resize(
    ST_AddBand(
      ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
      , 1, '8BUI', 255, 0
    )
    , 0.25, 0.9) AS rast
), bar AS (
  SELECT rid, ST_Metadata(rast) AS meta, rast FROM foo
)
SELECT rid, (meta).* FROM bar
```

rid	upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	
numbands										
1	0	0	500	500	1	-1	0	0	0	←
2	0	0	500	100	1	-1	0	0	0	←
3	0	0	250	900	1	-1	0	0	0	←

(3 rows)

例

[ST_Resample](#), [ST_Rescale](#), [ST_Reskew](#), [ST_SnapToGrid](#)

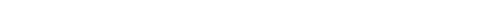
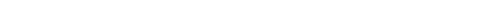
11.7.12 ST_Transform

ST Transform — 将具有指定 SRS 的几何体转换为具有指定 SRS 的几何体。最近邻、双线性、三次、三次样条、Lanczos 最近邻。最近邻最近邻。

Synopsis

```
raster ST_Transform(raster rast, integer srid, text algorithm=NearestNeighbor, double precision
maxerr=0.125, double precision scalex, double precision scaley);
raster ST_Transform(raster rast, integer srid, double precision scalex, double precision scaley, text
algorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_Transform(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision
maxerr=0.125);
```



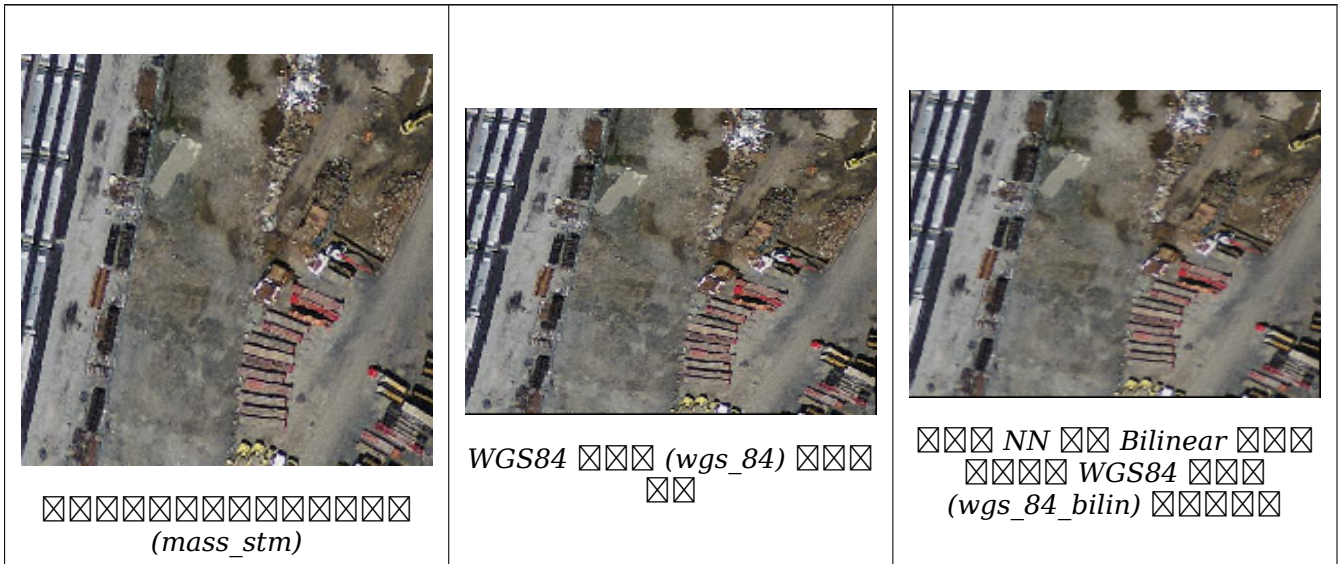
 (pixel warp) 
 NearestNeighbor 
 0.125 

GDAL Warp resampling methods: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', 'Lanczos'.

ST_Transform(*geom*) ST_SetSRID(*geom*, *srid*). ST_Transform(*geom*, *srid*)
 (*geom* is a geometry object, *srid* is a SRID value) *geom*, ST_SetSRID(*geom*,
srid) SRID *srid*.

XXXXXXXX, 3 alignto XXXXXXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXXXXXXXXXXXXXXX
XX (SRID) XXXXXXXXXXXX, (ST SameAlignment = TRUE XXX) XXXXXXXXXXXXXXXXXXXX.

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

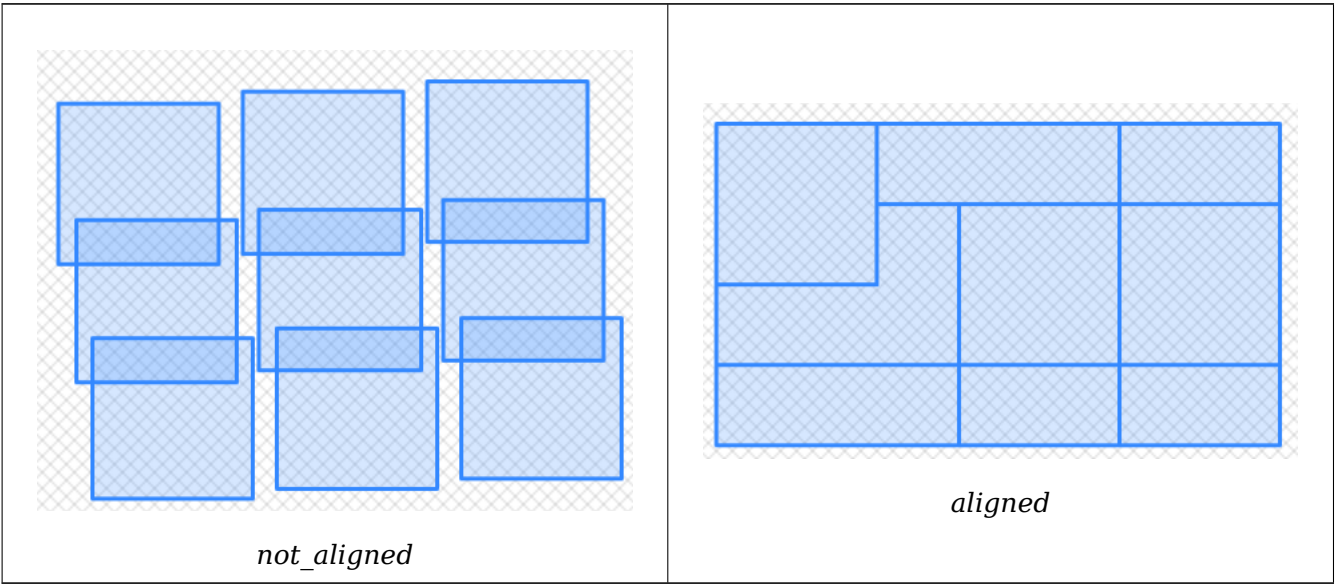
  SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599800, 100, -100, 0, 0, 2163), 1, '16BUI', 10, 0) AS rast UNION ALL
  SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599800, 100, -100, 0, 0, 2163), 1, '16BUI', 20, 0) AS rast UNION ALL
  SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 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 简体中文

11.8.1 ST_SetBandNoDataValue

ST_SetBandNoDataValue — NODATA 标识符。 标识符。 1 标识符。 标识符 NODATA 标识符, nodata value = NULL 标识符。

Synopsis

raster **ST_SetBandNoDataValue**(raster rast, double precision nodatavalue);
raster **ST_SetBandNoDataValue**(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);

¶¶

¶¶¶ NODATA 值。 1 值。 函数 `ST_Polygon`, `ST_DumpAsPolygons`, 以及 `ST_PixelAs...` 函数。

¶¶

```
-- change just first band no data value
UPDATE dummy_rast
  SET rast = ST_SetBandNoDataValue(rast,1, 254)
WHERE rid = 2;

-- change no data band value of bands 1,2,3
UPDATE dummy_rast
  SET rast =
    ST_SetBandNoDataValue(
      ST_SetBandNoDataValue(
        ST_SetBandNoDataValue(
          rast,1, 254)
        ,2,99),
      3,108)
  WHERE rid = 2;

-- wipe out the nodata value this will ensure all pixels are considered for all processing ↩
functions
UPDATE dummy_rast
  SET rast = ST_SetBandNoDataValue(rast,1, NULL)
WHERE rid = 2;
```

¶¶

`ST_BandNoDataValue`, `ST_NumBands`

11.8.2 ST_SetBandIsNoData

`ST_SetBandIsNoData` — 将指定带的值设置为 isnodata。

Synopsis

raster **ST_SetBandIsNoData**(raster rast, integer band=1);

¶¶

将指定带的值设置为 isnodata。 1 值。 函数 `ST_BandIsNoData` 函数。

2.0.0 版本。

实验4.1

```
-- Create dummy table with one raster column
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ←
  = 3.
-- In the second band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'4' -- hasnodatavalue set to true, isnodata value set to false (when it should be true)
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected false
select st_bandisnodata(rast, 1, TRUE) from dummy_rast where rid = 1; -- Expected true

-- The isnodata flag is dirty. We are going to set it to true
update dummy_rast set rast = st_setbandisnodata(rast, 1) where rid = 1;

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true
```

☒☒

[ST_BandNoDataValue](#), [ST_NumBands](#), [ST_SetBandNoDataValue](#), [ST_BandIsNoData](#)

11.8.3 ST_SetBandPath

`ST_SetBandPath` — Update the external path and band number of an out-db band

Synopsis

raster **ST_SetBandPath**(raster rast, integer band, text outdbpath, integer outdbindex, boolean force=false)

☒☒

Updates an out-db band's external raster file path and external band number.



Note

If force is set to true, no tests are done to ensure compatibility (e.g. alignment, pixel support) between the external raster file and the PostGIS raster. This mode is intended for file system changes where the external raster resides.

Availability: 2.5.0

☒☒

```
WITH foo AS (
    SELECT
        ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↵
        loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
    1 AS query,
    *
FROM ST_BandMetadata(
    (SELECT rast FROM foo),
    ARRAY[1,3,2]::int[]
)
UNION ALL
SELECT
    2,
    *
FROM ST_BandMetadata(
    (
        SELECT
            ST_SetBandPath(
                rast,
                2,
                '/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected2.tif ↵
                ',
                1
            ) AS rast
        FROM foo
    ),
    *
```



```
ARRAY[1,3,2]::int[]
)
ORDER BY 1, 2;

query | bandnum | pixeltypes | nodatavalue | isoutdb | path | outdbbandnum
-----+-----+-----+-----+-----+-----+-----
1 | 1 | 8BUI | | t | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif | 1
1 | 2 | 8BUI | | t | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif | 2
1 | 3 | 8BUI | | t | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif | 3
2 | 1 | 8BUI | | t | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif | 1
2 | 2 | 8BUI | | t | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected2.tif | 1
2 | 3 | 8BUI | | t | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif | 3
```

`ST_BandMetaData`, `ST_SetBandIndex`

11.8.4 `ST_SetBandIndex`

`ST_SetBandIndex` — Update the external band number of an out-db band

Synopsis

raster **`ST_SetBandIndex`**(raster rast, integer band, integer outdbindex, boolean force=false);

Updates an out-db band’s external band number. This does not touch the external raster file associated with the out-db band



Note
If force is set to true, no tests are done to ensure compatibility (e.g. alignment, pixel support) between the external raster file and the PostGIS raster. This mode is intended for where bands are moved around in the external raster file.



Note
Internally, this method replaces the PostGIS raster’s band at index band with a new band instead of updating the existing path information.

Availability: 2.5.0

❏

```
WITH foo AS (  
    SELECT  
        ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↵  
        loader/Projected.tif', NULL::int[]) AS rast  
)  
SELECT  
    1 AS query,  
    *  
FROM ST_BandMetadata(  
    (SELECT rast FROM foo),  
    ARRAY[1,3,2]::int[]  
)  
UNION ALL  
SELECT  
    2,  
    *  
FROM ST_BandMetadata(  
    (  
        SELECT  
            ST_SetBandIndex(  
                rast,  
                2,  
                1  
            ) AS rast  
        FROM foo  
    ),  
    ARRAY[1,3,2]::int[]  
)  
ORDER BY 1, 2;
```

query	bandnum	pixeltype	nodatavalue	isoutdb	path	
	outdbbandnum					
1	1	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵	
		raster/test/regress/loader/Projected.tif			1	
1	2	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵	
		raster/test/regress/loader/Projected.tif			2	
1	3	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵	
		raster/test/regress/loader/Projected.tif			3	
2	1	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵	
		raster/test/regress/loader/Projected.tif			1	
2	2	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵	
		raster/test/regress/loader/Projected.tif			1	
2	3	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵	
		raster/test/regress/loader/Projected.tif			3	

❏

ST_BandMetaData, ST_SetBandPath

11.9 统计函数

11.9.1 ST_Count

ST_Count — 返回栅格中指定值的像素数量。如果指定了 `exclude_nodata_value` 参数，则排除 NODATA 值的像素。默认情况下，`exclude_nodata_value` 为 `true`。

Synopsis

```
bigint ST_Count(raster rast, integer nband=1, boolean exclude_nodata_value=true);
bigint ST_Count(raster rast, boolean exclude_nodata_value);
```

参数

`nband` 指定要统计的波段。如果未指定，则统计所有波段。默认值为 1。



Note

`exclude_nodata_value` 参数，如果为 `true`，则排除 NODATA 值的像素。如果为 `false`，则包含 NODATA 值的像素。

2.2.0 版本中，`ST_Count(rastertable, rastercolumn, ...)` 函数被弃用。请使用 `ST_CountAgg` 函数。

2.0.0 版本中，`ST_Count` 函数被弃用。

示例

```
--example will count all pixels not 249 and one will count all pixels. --
SELECT rid, ST_Count(ST_SetBandNoDataValue(rast,249)) As exclude_nodata,
       ST_Count(ST_SetBandNoDataValue(rast,249),false) As include_nodata
FROM dummy_rast WHERE rid=2;
```

rid	exclude_nodata	include_nodata
2	23	25

相关链接

[ST_CountAgg](#), [ST_SummaryStats](#), [ST_SetBandNoDataValue](#)

11.9.2 ST_CountAgg

ST_CountAgg — 聚合函数，返回栅格中指定值的像素数量。如果指定了 `exclude_nodata_value` 参数，则排除 NODATA 值的像素。默认情况下，`exclude_nodata_value` 为 `true`。

Synopsis

```
bigint ST_CountAgg(raster rast, integer nband, boolean exclude_nodata_value, double precision
sample_percent);
bigint ST_CountAgg(raster rast, integer nband, boolean exclude_nodata_value);
bigint ST_CountAgg(raster rast, boolean exclude_nodata_value);
```

返回

返回一个 64 位整数。nband 默认为 1。exclude_nodata_value 默认为 true，nodata 值将被忽略。sample_percent 默认为 1，表示 100% 的样本。

2.2.0 版本引入。

示例

```
WITH foo AS (
  SELECT
    rast.rast
  FROM (
    SELECT ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_AddBand(
            ST_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,0)
            , 1, '64BF', 0, 0
          )
          , 1, 1, 1, -10
        )
        , 1, 5, 4, 0
      )
      , 1, 5, 5, 3.14159
    ) AS rast
  ) AS rast
  FULL JOIN (
    SELECT generate_series(1, 10) AS id
  ) AS id
  ON 1 = 1
)
SELECT
  ST_CountAgg(rast, 1, TRUE)
FROM foo;

 st_countagg
-----
          20
(1 row)
```

参见

[ST_Count](#), [ST_SummaryStats](#), [ST_SetBandNoDataValue](#)

11.9.3 ST_Histogram

ST_Histogram — (bin; 栅格, 精度, 是否排除空值, 是否右开) 返回包含 min, max, count, percent 的表。nband 默认为 1。

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

表包含 min, max, count, percent 列。nband 默认为 1。



Note

如果 exclude_nodata_value 为 true，则空值将被忽略。如果为 false，则空值将被计入 count。

width width: 精度/栅格单元大小。如果 width 为 NULL，则 width 默认为 1。

例如: width [a, b, c] 返回 [a, b, c, a, b, c, a, b, c]。

bins (breakout) 值: 返回的统计结果数。默认为 9。

right 是否右开 (true) 或左开 (false)。X 轴为 [a, b) 或 (a, b]。

Changed: 3.1.0 Removed ST_Histogram(table_name, column_name) variant.

2.0.0 版本引入。

注意: 1, 2, 3 是返回的统计结果数。

```
SELECT band, (stats).*
FROM (SELECT rid, band, ST_Histogram(rast, band) As stats
      FROM dummy_rast CROSS JOIN generate_series(1,3) As band
      WHERE rid=2) As foo;
```

band	min	max	count	percent
1	249	250	2	0.08
1	250	251	2	0.08
1	251	252	1	0.04
1	252	253	2	0.08
1	253	254	18	0.72

2	78	113.2	11	0.44
2	113.2	148.4	4	0.16
2	148.4	183.6	4	0.16
2	183.6	218.8	1	0.04
2	218.8	254	5	0.2
3	62	100.4	11	0.44
3	100.4	138.8	5	0.2
3	138.8	177.2	4	0.16
3	177.2	215.6	1	0.04
3	215.6	254	4	0.16

图 11.9.3: 使用 ST_Histogram 函数生成直方图

```
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)

图 11.9.4

ST_Count, ST_SummaryStats, ST_SummaryStatsAgg

11.9.4 ST_Quantile

ST_Quantile — 返回指定百分位 (population) 的量化值 (quantile) 的函数。例如，返回 25%、50%、75% 的百分位 (percentile) 的函数。

Synopsis

SETOF record **ST_Quantile**(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] quantiles=NULL);
 SETOF record **ST_Quantile**(raster rast, double precision[] quantiles);
 SETOF record **ST_Quantile**(raster rast, integer nband, double precision[] quantiles);
 double precision **ST_Quantile**(raster rast, double precision quantile);
 double precision **ST_Quantile**(raster rast, boolean exclude_nodata_value, double precision quantile=NULL);
 double precision **ST_Quantile**(raster rast, integer nband, double precision quantile);
 double precision **ST_Quantile**(raster rast, integer nband, boolean exclude_nodata_value, double precision quantile);
 double precision **ST_Quantile**(raster rast, integer nband, double precision quantile);

注意

ST_Quantile (population) ST_Quantile (quantile) 返回一个数组，包含 25%, 50%, 75% 分位数 (percentile) 的值。



Note

exclude_nodata_value 默认为 true，NODATA 值将被排除。

Changed: 3.1.0 Removed ST_Quantile(table_name, column_name) variant.

2.0.0 版本引入。

注意

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Example will consider only pixels of band 1 that are not 249 and in named quantiles --
```

```
SELECT (pvq).*
FROM (SELECT ST_Quantile(rast, ARRAY[0.25,0.75]) As pvq
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvq).quantile;
```

```
quantile | value
-----+-----
0.25    | 253
0.75    | 254
```

```
SELECT ST_Quantile(rast, 0.75) As value
FROM dummy_rast WHERE rid=2;
```

```
value
-----
254
```

```
--real live example. Quantile of all pixels in band 2 intersecting a geometry
SELECT rid, (ST_Quantile(rast,2)).* As pvc
FROM o_4_boston
WHERE ST_Intersects(rast,
  ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706
    892151,224486 892151))',26986) ←
```

```
)  
ORDER BY value, quantile, rid  
;
```

rid	quantile	value
1	0	0
2	0	0
14	0	1
15	0	2
14	0.25	37
1	0.25	42
15	0.25	47
2	0.25	50
14	0.5	56
1	0.5	64
15	0.5	66
2	0.5	77
14	0.75	81
15	0.75	87
1	0.75	94
2	0.75	106
14	1	199
1	1	244
2	1	255
15	1	255



ST_Count, ST_SummaryStats, ST_SummaryStatsAgg, ST_SetBandNoDataValue

11.9.5 ST_SummaryStats

ST_SummaryStats — 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);
```



count, sum, mean, stddev, min, max **summarystats**. **nbands** 1 **dtype**.



Note!

Note
 ☐☐☐☐☐ nodata ☐☐☐☐☐☐☐☐☐☐☐☐☐☐. ☐☐☐☐☐☐☐☐☐☐☐☐☐☐
 exclude nodata_value ☐☐☐☐☐☐☐☐☐☐☐☐.

**Note**

ST_SummaryStats 函数返回一个表，其中包含统计信息。sample_percent 参数默认为 1，表示 100% 的样本。

2.2.0 版本中，ST_SummaryStats(rastertable, rastercolumn, ...) 函数返回一个表。ST_SummaryStatsAgg 函数返回一个聚合函数。

2.0.0 版本中，ST_SummaryStats 函数返回一个表。

例：统计栅格数据

```
SELECT rid, band, (stats).*
FROM (SELECT rid, band, ST_SummaryStats(rast, band) As stats
      FROM dummy_rast CROSS JOIN generate_series(1,3) As band
      WHERE rid=2) As foo;
```

rid	band	count	sum	mean	stddev	min	max
2	1	23	5821	253.086957	1.248061	250	254
2	2	25	3682	147.28	59.862188	78	254
2	3	25	3290	131.6	61.647384	62	254

例：统计栅格数据的统计信息

PostGIS 版本 64 位的栅格数据，其统计信息（例如 102,000 个像素，150x150 像素的栅格，134,000 个像素）的统计信息为 574 个统计信息。

```
WITH
-- our features of interest
feat AS (SELECT gid As building_id, geom_26986 As geom FROM buildings AS b
        WHERE gid IN(100, 103,150)
        ),
-- clip band 2 of raster tiles to boundaries of builds
-- then get stats for these clipped regions
b_stats AS
(SELECT building_id, (stats).*
FROM (SELECT building_id, ST_SummaryStats(ST_Clip(rast,2,geom)) As stats
      FROM aerials.boston
      INNER JOIN feat
      ON ST_Intersects(feat.geom,rast)
      ) As foo
      )
-- finally summarize stats
SELECT building_id, SUM(count) As num_pixels
      , MIN(min) As min_pval
      , MAX(max) As max_pval
      , SUM(mean*count)/SUM(count) As avg_pval
      FROM b_stats
      WHERE count
      > 0
      GROUP BY building_id
      ORDER BY building_id;
```

building_id	num_pixels	min_pval	max_pval	avg_pval
100	1090	1	255	61.0697247706422
103	655	7	182	70.5038167938931
150	895	2	252	185.642458100559

例: 统计每个波段

```
-- stats for each band --
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band) As stats
      FROM generate_series(1,3) As band) As foo;
```

band	count	sum	mean	stddev	min	max
1	8450000	725799	82.7064349112426	45.6800222638537	0	255
2	8450000	700487	81.4197705325444	44.2161184161765	0	255
3	8450000	575943	74.682739408284	44.2143885481407	0	255

```
-- For a table -- will get better speed if set sampling to less than 100%
-- Here we set to 25% and get a much faster answer
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band,true,0.25) As stats
      FROM generate_series(1,3) As band) As foo;
```

band	count	sum	mean	stddev	min	max
1	2112500	180686	82.6890480473373	45.6961043857248	0	255
2	2112500	174571	81.448503668639	44.2252623171821	0	255
3	2112500	144364	74.6765884023669	44.2014869384578	0	255

例

summarystats, **ST_SummaryStatsAgg**, **ST_Count**, **ST_Clip**

11.9.6 ST_SummaryStatsAgg

ST_SummaryStatsAgg — 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);
```

例

统计每个波段的 count, sum, mean, stddev, min, max 使用 **summarystats** 函数。指定 nband 为 1 表示所有波段。



Note

如果 raster 包含 nodata 值，则使用 exclude_nodata_value 参数。如果 exclude_nodata_value 为 true，则统计将忽略 nodata 值。



Note

ST_SummaryStatsAgg 函数支持 sample_percent 参数，其值在 0 到 1 之间，表示对输入数据进行随机采样。默认值为 1.0，即使用全部数据。

2.2.0 新增功能。

SQL

```
WITH foo AS (
  SELECT
    rast.rast
  FROM (
    SELECT ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_AddBand(
            ST_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,0)
            , 1, '64BF', 0, 0
          )
          , 1, 1, 1, -10
        )
        , 1, 5, 4, 0
      )
      , 1, 5, 5, 3.14159
    ) AS rast
  ) AS rast
  FULL JOIN (
    SELECT generate_series(1, 10) AS id
  ) AS id
  ON 1 = 1
)
SELECT
  (stats).count,
  round((stats).sum::numeric, 3),
  round((stats).mean::numeric, 3),
  round((stats).stddev::numeric, 3),
  round((stats).min::numeric, 3),
  round((stats).max::numeric, 3)
FROM (
  SELECT
    ST_SummaryStatsAgg(rast, 1, TRUE, 1) AS stats
  FROM foo
) bar;
```

count	round	round	round	round	round
20	-68.584	-3.429	6.571	-10.000	3.142

(1 row)

SQL

[summarystats](#), [ST_SummaryStats](#), [ST_Count](#), [ST_Clip](#)

11.9.7 ST_ValueCount

ST_ValueCount — 返回指定栅格中指定值的统计信息 (聚合函数) 的表。返回的表包含两列: 值 (value) 和计数 (count)。如果指定了 `exclude_nodata_value` 参数, 则 NODATA 值将被排除在统计之外。如果指定了 `roundto` 参数, 则计数将四舍五入到指定的精度。

Synopsis

```
SETOF record ST_ValueCount(raster rast, integer nband=1, boolean exclude_nodata_value=true,
double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, in-
teger OUT count);
SETOF record ST_ValueCount(raster rast, integer nband, double precision[] searchvalues, double
precision roundto=0, double precision OUT value, integer OUT count);
SETOF record ST_ValueCount(raster rast, double precision[] searchvalues, double precision roundto=0,
double precision OUT value, integer OUT count);
bigint ST_ValueCount(raster rast, double precision searchvalue, double precision roundto=0);
bigint ST_ValueCount(raster rast, integer nband, boolean exclude_nodata_value, double precision
searchvalue, double precision roundto=0);
bigint ST_ValueCount(raster rast, integer nband, double precision searchvalue, double precision
roundto=0);
SETOF record ST_ValueCount(text rastertable, text rastercolumn, integer nband=1, boolean ex-
clude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, dou-
ble precision OUT value, integer OUT count);
SETOF record ST_ValueCount(text rastertable, text rastercolumn, double precision[] searchvalues,
double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record ST_ValueCount(text rastertable, text rastercolumn, integer nband, double precision[]
searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
bigint ST_ValueCount(text rastertable, text rastercolumn, integer nband, boolean exclude_nodata_value,
double precision searchvalue, double precision roundto=0);
bigint ST_ValueCount(text rastertable, text rastercolumn, double precision searchvalue, double pre-
cision roundto=0);
bigint ST_ValueCount(text rastertable, text rastercolumn, integer nband, double precision search-
value, double precision roundto=0);
```

返回

返回的表包含两列: 值 (value) 和计数 (count)。如果指定了 `exclude_nodata_value` 参数, 则 NODATA 值将被排除在统计之外。

如果指定了 `nband` 参数, 则只统计指定波段中的值。如果指定了 `searchvalues` 参数, 则只统计指定值。如果指定了 `roundto` 参数, 则计数将四舍五入到指定的精度。



Note

`exclude_nodata_value` 参数, NODATA 值将被排除在统计之外。

2.0.0 版本引入。

返回

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Example will count only pixels of band 1 that are not 249. --
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast) As pvc
      FROM dummy_rast WHERE rid=2) As foo
      ORDER BY (pvc).value;
```

value	count
250	2
251	1
252	2
253	6
254	12

```
-- Example will count all pixels of band 1 including 249 --
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,1,false) As pvc
      FROM dummy_rast WHERE rid=2) As foo
      ORDER BY (pvc).value;
```

value	count
249	2
250	2
251	1
252	2
253	6
254	12

```
-- Example will count only non-nodata value pixels of band 2
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,2) As pvc
      FROM dummy_rast WHERE rid=2) As foo
      ORDER BY (pvc).value;
```

value	count
78	1
79	1
88	1
89	1
96	1
97	1
98	1
99	2
112	2

```
;
```

```
--real live example. Count all the pixels in an aerial raster tile band 2 intersecting a geometry ↵
```

```
-- and return only the pixel band values that have a count > 500
```

```
SELECT (pvc).value, SUM((pvc).count) As total
FROM (SELECT ST_ValueCount(rast,2) As pvc
      FROM o_4_boston
      WHERE ST_Intersects(rast,
        ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 ↵
        892151,224486 892151))',26986)
      )
      ) As foo
```



```
)
)).* AS metadata;

upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  ↔
numbands
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
          0.5 |          0.5 |    10 |    20 |     2 |     3 |     0 |     0 |    10 |  ↔
          0
```



ST_MetaData, ST_RastFromHexWKB, ST_AsBinary/ST_AsWKB, ST_AsHexWKB

11.10.2 ST_RastFromHexWKB

ST_RastFromHexWKB — Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.

Synopsis

```
raster ST_RastFromHexWKB(text wkb);
```



Given a Well-Known Binary (WKB) raster in Hex representation, return a raster.

Availability: 2.5.0



```
SELECT (ST_Metadata(
  ST_RastFromHexWKB(
    '0100000000000000000000000040000000000000008400000000000000000 ↵
      E03F00000000000000E03F0000000000000000000000000000000000A0000000A001400'
  )
)).* AS metadata;
```

upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	↵
0.5	0.5	10	20	2	3	0	0	10	↵



ST MetaData, ST RastFromWKB, ST AsBinary/ST AsWKB, ST AsHexWKB

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 国际化

```
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;
```

国际化

Section [10.3, ST_GDALDrivers, ST_SRID](#)

11.11.4 ST_AsJPEG

ST_AsJPEG — 国际化 JPEG (Joint Photographic Exports Group) 国际化 (国际化 国际化) 国际化。国际化, 国际化 1 国际化 3 国际化。国际化。国际化 3 国际化 3 国际化 RGB 国际化。

Synopsis

```
bytea ST_AsJPEG(raster rast, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer nband, integer quality);
bytea ST_AsJPEG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, integer quality);
```

描述

ST_AsJPEG(raster rast, text[] options=NULL) 使用 JPEG (Joint Photographic Experts Group) 格式将栅格数据导出为字节数组。该函数使用 ST_AsGDALRaster 函数将栅格数据导出为 GDAL 格式。默认情况下，输出 1 个波段，3 个波段。输出 3 个波段，3 个波段。输出 3 个波段，3 个波段。

- **nband** - 输出单波段栅格。
- **nbands** - 输出多波段栅格 (JPEG 支持 3 个波段)。输出 RGB 格式。输出 ARRAY[3,2,1] 格式 3 个波段，输出 2 个波段，输出 1 个波段。
- **quality** - 1 到 100 之间的整数。输出 75 到 100 之间的整数。
- **options** - JPEG 格式使用 GDAL 驱动程序 (ST_GDALDrivers 函数) 创建选项 (create_options 函数)。JPEG 格式，输出 PROGRESSIVE ON/OFF 格式 75 到 100 之间的整数。输出 QUALITY 格式。输出 GDAL 格式。

2.0.0 版本。GDAL 1.6.0 版本。

示例：

```
-- output first 3 bands 75% quality
SELECT ST_AsJPEG(rast) As rastjpg
FROM dummy_rast WHERE rid=2;

-- output only first band as 90% quality
SELECT ST_AsJPEG(rast,1,90) As rastjpg
FROM dummy_rast WHERE rid=2;

-- output first 3 bands (but make band 2 Red, band 1 green, and band 3 blue, progressive and 90% quality
SELECT ST_AsJPEG(rast,ARRAY[2,1,3],ARRAY['QUALITY=90','PROGRESSIVE=ON']) As rastjpg
FROM dummy_rast WHERE rid=2;
```

描述

Section 10.3, ST_GDALDrivers, ST_AsGDALRaster, ST_AsPNG, ST_AsTIFF

11.11.5 ST_AsPNG

ST_AsPNG — 使用 PNG (Portable Network Graphics) 格式将栅格数据导出为字节数组。该函数使用 ST_AsGDALRaster 函数将栅格数据导出为 GDAL 格式。默认情况下，输出 1 个，3 个，4 个波段。输出 2 个，4 个波段。输出 1 个波段。输出 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);
```

描述

ST_AsPNG(Portable Network Graphics) 函数。ST_AsGDALRaster 函数。返回 3 个字节。srid 返回 SRID 值。返回类型如下：

- nband - 返回 nband 值。
- nbands - 返回 nbands (PNG 支持 4 个通道)。返回 RGBA 值。返回 ARRAY[3,2,1] 3 个字节, 2 个字节, 1 个字节。
- compression - 1 到 9 的压缩值。返回压缩值。
- options - PNG 选项 GDAL 选项 (ST_GDALDrivers 返回 PNG 选项 create_options 值)。PNG 选项, 返回 ZLEVEL(压缩级别 - 默认 6) 值, 返回 ARRAY['ZLEVEL=9'] 值。返回 2 个字节。返回 GDAL 选项值。

2.0.0 版本。GDAL 1.6.0 版本。

示例

```
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;
```

参见

ST_AsGDALRaster, ST_ColorMap, ST_GDALDrivers, Section 10.3

11.11.6 ST_AsTIFF

ST_AsTIFF — Return the raster selected bands as a single TIFF image (byte array). If no band is specified or any of specified bands does not exist in the raster, then will try to use all bands.

Synopsis

```
bytea ST_AsTIFF(raster rast, text[] options="", integer srid=sameasource);
bytea ST_AsTIFF(raster rast, text compression="", integer srid=sameasource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text compression="", integer srid=sameasource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text[] options, integer srid=sameasource);
```

描述

ST_AsTIFF 将栅格数据转换为 TIFF (Tagged Image File Format) 格式。ST_AsTIFF 函数接受一个栅格值，并返回一个文本值，该值表示栅格数据的 TIFF 格式。ST_AsTIFF 函数使用 ST_AsGDALRaster 函数将栅格数据转换为 GDAL 格式。ST_AsTIFF 函数使用 SRS 信息，并返回一个文本值。ST_AsTIFF 函数的语法如下：

- **nbands** - 指定要输出的波段数 (PNG 支持 3 个波段)。如果未指定，则使用 RGB 波段。如果指定了 ARRAY[3,2,1]，则输出 3 个波段，第 2 个波段，第 1 个波段。
- **compression** - 指定压缩格式：JPEG90(有损压缩), LZW, JPEG, DEFLATE9
- **options** - GTiff 的 GDAL 选项 (ST_GDALDrivers 中的 GTiff 选项 create_options 中的选项)。GDAL 选项在 GDAL 文档中定义。
- **srid** - 指定 spatial_ref_sys 的 SRID 值。如果未指定，则使用默认值。

2.0.0 版本引入。GDAL 1.6.0 版本引入。

示例: JPEG 90%

```
SELECT ST_AsTIFF(rast, 'JPEG90') As rasttiff
FROM dummy_rast WHERE rid=2;
```

参数

ST_GDALDrivers, ST_AsGDALRaster, ST_SRID

11.12 几何函数

11.12.1 ST_Clip

ST_Clip — Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.

Synopsis

```
raster ST_Clip(raster rast, integer[] nband, geometry geom, double precision[] nodataval=NULL,
boolean crop=TRUE, boolean touched=FALSE);
raster ST_Clip(raster rast, integer nband, geometry geom, double precision nodataval, boolean crop=TRUE,
boolean touched=FALSE);
raster ST_Clip(raster rast, integer nband, geometry geom, boolean crop, boolean touched=FALSE);
raster ST_Clip(raster rast, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE,
boolean touched=FALSE);
raster ST_Clip(raster rast, geometry geom, double precision nodataval, boolean crop=TRUE, boolean
touched=FALSE);
raster ST_Clip(raster rast, geometry geom, boolean crop, boolean touched=FALSE);
```

geom

geom 是几何对象。如果 geom 是 NULL，则返回 NULL。

ST_Clip 返回与 geom 相交的 rast 的副本。如果 rast 包含 NODATA 值，则返回的副本也包含 NODATA 值。NODATA 值在 rast 中被替换为 ST_MinPossibleValue(ST_Band(rast, 1))。如果 rast 包含 NODATA 值，则返回的副本也包含 NODATA 值。NODATA 值在 rast 中被替换为 ST_MinPossibleValue(ST_Band(rast, 1))。如果 rast 包含 NODATA 值，则返回的副本也包含 NODATA 值。NODATA 值在 rast 中被替换为 ST_MinPossibleValue(ST_Band(rast, 1))。

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 引入。

更新: 2.1.0 引入 C 语言支持。

Examples here use Massachusetts aerial data available on MassGIS site [MassGIS Aerial Orthos](#).

Examples: Comparing selecting all touched vs. not all touched

```
SELECT ST_Count(rast) AS count_pixels_in_orig, ST_Count(rast_touched) AS all_touched_pixels ←
, ST_Count(rast_not_touched) AS default_clip
FROM ST_AsRaster(ST_Letters('R'), scalex =
> 1.0, scaley =
> -1.0) AS r(rast)
INNER JOIN ST_GeomFromText('LINESTRING(0 1, 5 6, 10 10)') AS g(geom)
ON ST_Intersects(r.rast,g.geom)
, ST_Clip(r.rast, g.geom, touched =
> true) AS rast_touched
, ST_Clip(r.rast, g.geom, touched =
> false) AS rast_not_touched;
```

count_pixels_in_orig	all_touched_pixels	default_clip
2605	16	10

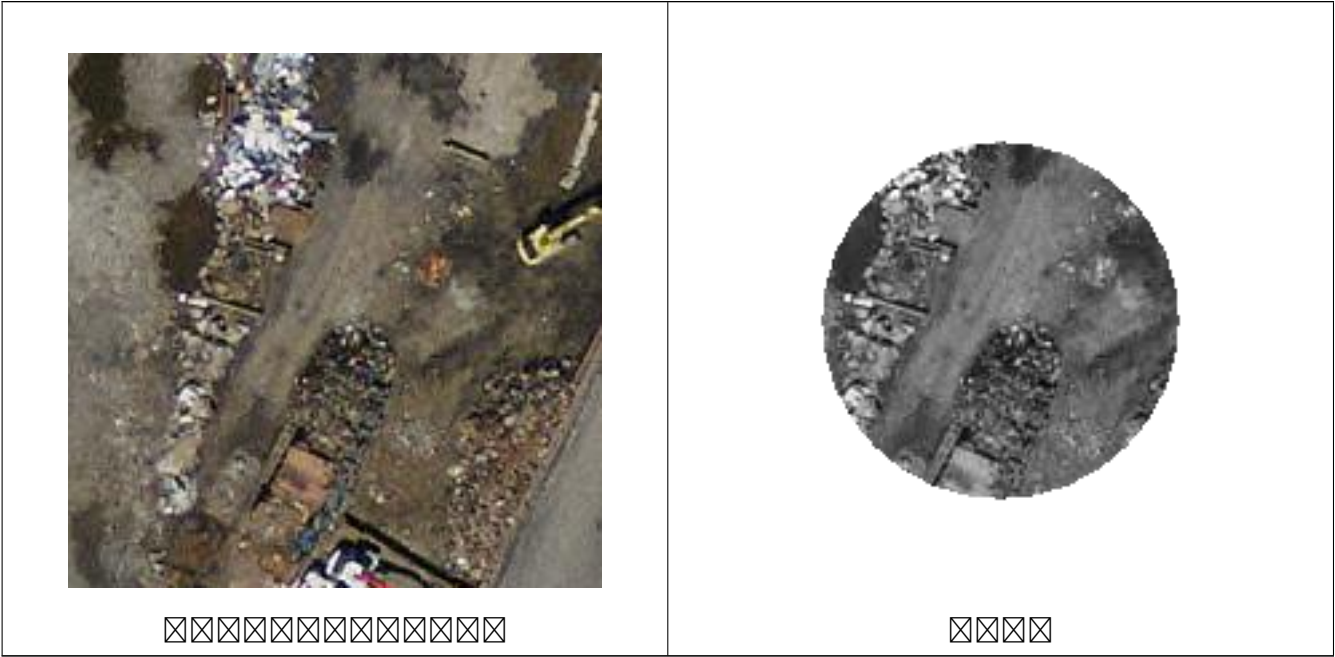
(1 row)

Examples: 1 band clipping (not touched)

```
-- Clip the first band of an aerial tile by a 20 meter buffer.
SELECT ST_Clip(rast, 1,
, ST_Buffer(ST_Centroid(ST_Envelope(rast)),20)
) from aerials.boston
WHERE rid = 4;
```

```
-- Demonstrate effect of crop on final dimensions of raster
-- Note how final extent is clipped to that of the geometry
-- if crop = true
SELECT ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, true))) As xmax_w_trim,
       ST_XMax(clipper) As xmax_clipper,
       ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, false))) As xmax_wo_trim,
       ST_XMax(ST_Envelope(rast)) As xmax_rast_orig
FROM (SELECT rast, ST_Buffer(ST_Centroid(ST_Envelope(rast)),6) As clipper
      FROM aerials.boston
      WHERE rid = 6) As foo;
```

xmax_w_trim	xmax_clipper	xmax_wo_trim	xmax_rast_orig
230657.436173996	230657.436173996	230666.436173996	230666.436173996



注意: `crop` 参数为 `1` 时，裁剪后的栅格尺寸将与几何图形的尺寸一致。

```
-- Same example as before, but we need to set crop to false to be able to use ST_AddBand
-- because ST_AddBand requires all bands be the same Width and height
SELECT ST_AddBand(ST_Clip(rast, 1,
                          ST_Buffer(ST_Centroid(ST_Envelope(rast)),20),false),
                  ARRAY[ST_Band(rast,2),ST_Band(rast,3)] ) from aerials.boston
WHERE rid = 6;
```



SQL: 缓冲

```
-- Clip all bands of an aerial tile by a 20 meter buffer.  
-- Only difference is we don't specify a specific band to clip  
-- so all bands are clipped  
SELECT ST_Clipping(rast,  
    ST_Buffer(ST_Centroid(ST_Envelope(rast)), 20),  
    false  
    ) from aerials.boston  
WHERE rid = 4;
```



SQL

ST_AddBand, ST_Count, ST_MapAlgebra (callback function version), ST_Intersection

11.12.2 ST_ColorMap

ST_ColorMap — 8BUI (grayscale, RGB, RGBA) 4 1

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);
```



```
rast %>% nband %>% colmap %>% 8BUI %>% 4 %>%  
colmap %>% 8BUI
```

nband ☐☐☐☐☐☐☐☐☐, ☐☐ 1 ☐☐☐☐☐☐.

```

% colormap 16 colors.
colormap(16);

```

```
colormap :
```

- grayscale  greyscale - 8BUI  (shades of gray) 
- pseudocolor - 8BUI(RGBA)  4                                   
- fire - 8BUI(RGBA)  4                                   
- bluered - 8BUI(RGBA)  4                                   


[illegible]

```
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 <http://colorbrewer2.org/>.



Warning
ST_SetBandNoDataValue 函数中 NODATA 参数必须为 NULL。 NODATA 参数必须为 NULL。
ST_SetBandNoDataValue 函数中 NODATA 参数必须为 NULL。

2.1.0 测试数据。

SQL

创建测试数据。

```
-- 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;
```

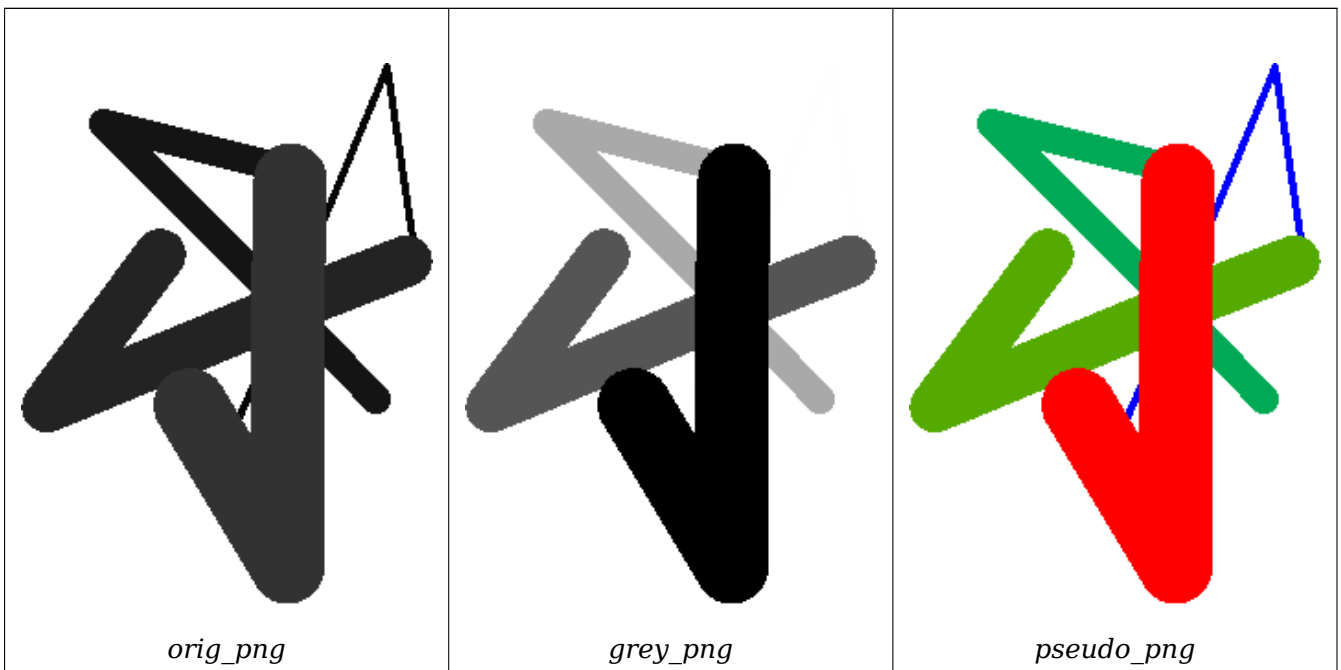
n_orig	ngrey	npseudo	nfire	nbluered	nred
1	1	4	4	4	3

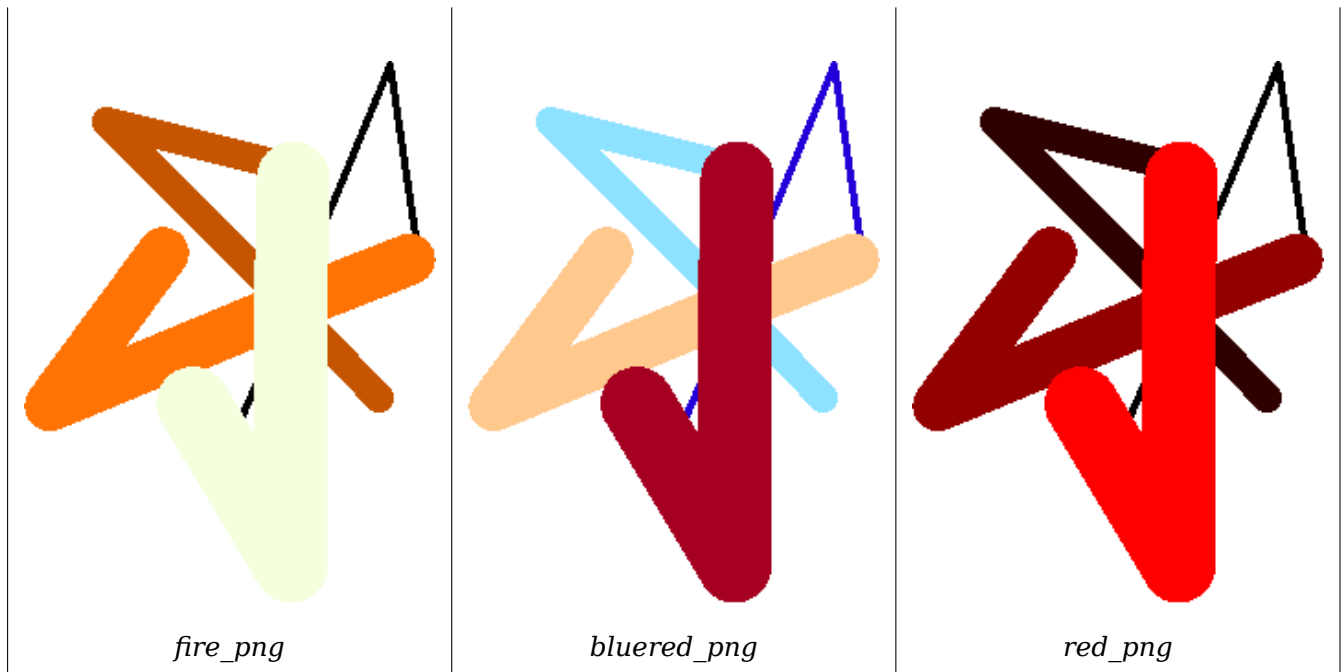
函数: ST_AsPNG 将 PostGIS 几何体转换为 PNG 图像

```

SELECT
  ST_AsPNG(rast) As orig_png,
  ST_AsPNG(ST_ColorMap(rast,1,'greyscale')) As grey_png,
  ST_AsPNG(ST_ColorMap(rast,1, 'pseudocolor')) As pseudo_png,
  ST_AsPNG(ST_ColorMap(rast,1, 'nfire')) As fire_png,
  ST_AsPNG(ST_ColorMap(rast,1, 'bluered')) As bluered_png,
  ST_AsPNG(ST_ColorMap(rast,1, '
100% 255  0  0
80%  160  0  0
50%  130  0  0
30%  30  0  0
20%  60  0  0
0%   0  0  0
nv 255 255 255
')) As red_png
FROM funky_shapes;

```





☒☒

[ST_AsPNG](#), [ST_AsRaster](#) [ST_MapAlgebra](#) (callback function version), [ST_Grayscale](#) [ST_NumBands](#), [ST_Reclass](#), [ST_SetBandNoDataValue](#), [ST_Union](#)

11.12.3 ST_Grayscale

ST_Grayscale — Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue

Synopsis

- (1) raster **ST_Grayscale**(raster rast, integer redband=1, integer greenband=2, integer blueband=3, text extenttype=INTERSECTION);
- (2) raster **ST_Grayscale**(rastbandarg[] rastbandargset, text extenttype=INTERSECTION);

☒☒

Create a raster with one 8BUI band given three input bands (from one or more rasters). Any input band whose pixel type is not 8BUI will be reclassified using [ST_Reclass](#).



Note

This function is not like [ST_ColorMap](#) with the grayscale keyword as [ST_ColorMap](#) operates on only one band while this function expects three bands for RGB. This function applies the following equation for converting RGB to Grayscale: $0.2989 * RED + 0.5870 * GREEN + 0.1140 * BLUE$

Availability: 2.5.0

实验: 实验 1

```

SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
  SELECT ST_AddBand(
    ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
    '/tmp/apple.png'::text,
    NULL::int[]
  ) AS rast
)
SELECT
  ST_AsPNG(rast) AS original_png,
  ST_AsPNG(ST_Grayscale(rast)) AS grayscale_png
FROM apple;

```

*original_png**grayscale_png*

实验: 实验 2

```

SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
  SELECT ST_AddBand(
    ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
    '/tmp/apple.png'::text,
    NULL::int[]
  ) AS rast
)
SELECT
  ST_AsPNG(rast) AS original_png,
  ST_AsPNG(ST_Grayscale(
    ARRAY[
      ROW(rast, 1)::rastbandarg, -- red
      ROW(rast, 2)::rastbandarg, -- green
      ROW(rast, 3)::rastbandarg, -- blue
    ]::rastbandarg[]
  )) AS grayscale_png

```

FROM apple;



ST_AsPNG, ST_Reclass, ST_ColorMap

11.12.4 ST_Intersection

ST Intersection — $\{x \in \mathbb{R}^n \mid x \in S \text{ and } x \in T\}$, $S \cap T$.
 $\{x \in \mathbb{R}^n \mid x \in S \text{ and } x \in T\}$.

Synopsis

```
setof geomval ST_Intersection(geometry geom, raster rast, integer band_num=1);
setof geomval ST_Intersection(raster rast, geometry geom);
setof geomval ST_Intersection(raster rast, integer band, geometry geomin);
raster ST_Intersection(raster rast1, raster rast2, double precision[] nodataval);
raster ST_Intersection(raster rast1, raster rast2, text returnband, double precision[] nodataval);
raster ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, double precision[]
nodataval);
raster ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, text returnband,
double precision[] nodataval);
```

[illegible]

```
geomval ST_DumpAsPolygon geomval). ST_Intersection(geometry, geometry) PostGIS  
ST_IsValid(geometry) NODATA WHERE ST_Intersect
```

```
geom_val = geom_val'.geom' val = val'.geom'
$: (ST_Intersection(rast, geom)).geom
```

`ST_MapAlgebraExpr`

```

XXXXXXXXXXXXXXXXXXXXX.XXXXXXXXXXX returnband XXXXXXXX
XXXXXXXXXXXX' BAND1', 'BAND2' XXX' BOTH' XXXXXXXXXX. XXXXXXX NODATA XXX
XXXXXXXXXXXXXXXXXXXXX NODATA XXXXXXXXXX. X, NODATA XXXXXXXXXXXXXXXXXXXX
NODATA XXXXXXXXXXXXXXXX.

```

```
ST_Intersection(geom1, geom2) NODATA.
ST_Band1, 'BAND2' 'BOTH' 1 2 NODATA.
nodaval[] NODATA.
NODATA, NODATA.
NODATA NODATA, ST_MinPossibleValue
NODATA. NODATA.

```

1. **ST Clip**



Note
如果输入是 NODATA 值，则返回 NODATA。ST_MapAlgebraExpr 函数用于处理这种情况。



Note
ST_Clip 函数用于修剪几何体。ST_Clip 函数返回修剪后的几何体。ST_Clip 函数可以用于修剪多边形、线、点等几何体。



Note
ST_Intersects 函数用于检查两个几何体是否相交。ST_Intersection 函数用于返回两个几何体的交集。

PostGIS 2.0.0 版本中，ST_Intersects 函数返回布尔值。2.0.0 版本中，ST_Intersection 函数返回几何体。

示例：查询相交的几何体

```
SELECT
  foo.rid,
  foo.gid,
  ST_AsText((foo.geomval).geom) As geomwkt,
  (foo.geomval).val
FROM (
  SELECT
    A.rid,
    g.gid,
    ST_Intersection(A.rast, g.geom) As geomval
  FROM dummy_rast AS A
  CROSS JOIN (
    VALUES
      (1, ST_Point(3427928, 5793243.85) ),
      (2, ST_GeomFromText('LINESTRING(3427927.85 5793243.75,3427927.8 5793243.75,3427927.8 5793243.8)')),
      (3, ST_GeomFromText('LINESTRING(1 2, 3 4)'))
  ) As g(gid,geom)
  WHERE A.rid = 2
) As foo;
```

rid	gid	geomwkt	val
2	1	POINT(3427928 5793243.85)	249
2	1	POINT(3427928 5793243.85)	253
2	2	POINT(3427927.85 5793243.75)	254
2	2	POINT(3427927.8 5793243.8)	251
2	2	POINT(3427927.8 5793243.8)	253
2	2	LINESTRING(3427927.8 5793243.75,3427927.8 5793243.8)	252
2	2	MULTILINESTRING((3427927.8 5793243.8,3427927.8 5793243.75),...)	250
2	3	GEOMETRYCOLLECTION EMPTY	

结果

geomval, ST_Intersects, ST_MapAlgebraExpr, ST_Clip, ST_AsText

11.12.5 ST_MapAlgebra (callback function version)

ST_MapAlgebra (callback function version) — 2次元ラスタ - 2次元 1 ラスタ, 2次元ラスタ, 2次元ラスタ 2次元ラスタ 1次元ラスタ 1次元ラスタ 1次元ラスタ 1次元ラスタ.

Synopsis

```
raster ST_MapAlgebra(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixelpixeltype=NULL,
text extenttype=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer[] nband, regprocedure callbackfunc, text pixelpixeltype=NULL,
text extenttype=FIRST, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, text pixelpixeltype=NULL,
text extenttype=FIRST, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast1, integer nband1, raster rast2, integer nband2, regprocedure callbackfunc, text pixelpixeltype=NULL, text extenttype=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, float8[] mask, boolean weighted, text pixelpixeltype=NULL, text extenttype=INTERSECTION, raster customextent=NULL, text[] VARIADIC userargs=NULL);
```

パラメータ

2次元 1 ラスタ, 2次元ラスタ, 2次元ラスタ 2次元ラスタ 1次元ラスタ 1次元ラスタ 1次元ラスタ 1次元ラスタ.

rast, rast1, rast2, rastbandargset 2次元 (代数) 2次元ラスタ

rastbandargset 2次元ラスタ/2次元ラスタ 2次元ラスタ 2次元ラスタ 2次元ラスタ 2次元ラスタ 2次元ラスタ 2次元ラスタ. 2次元 1次元ラスタ.

nband, nband1, nband2 2次元ラスタ. **nband** 2次元ラスタ 2次元ラスタ 2次元ラスタ 2次元ラスタ. **nband1** rast1 2次元ラスタ **nband2** 2次元ラスタ 2次元ラスタ 2次元ラスタ rast2 2次元ラスタ.

callbackfunc The callbackfunc parameter must be the name and signature of an SQL or PL/pgSQL function, cast to a regprocedure. An example PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION sample_callbackfunc(value double precision[][][], position integer[][], VARIADIC userargs text[])
RETURNS double precision
AS $$
BEGIN
    RETURN 0;
END;
$$ LANGUAGE 'plpgsql' IMMUTABLE;
```

The callbackfunc must have three arguments: a 3-dimension double precision array, a 2-dimension integer array and a variadic 1-dimension text array. The first argument value is the set of values (as double precision) from all input rasters. The three dimensions (where indexes are 1-based) are: raster #, row y, column x. The second argument position is the set of pixel positions from the output raster and input rasters. The outer dimension (where indexes are 0-based) is the raster #. The position at outer dimension index 0 is the output raster's pixel position. For each outer dimension, there are two elements in the inner dimension for X and Y. The third argument userargs is for passing through any user-specified arguments.

Passing a regprocedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'sample callbackfunc(double precision[], integer[], text[])':regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a `regprocedure`.

mask An n-dimensional array (matrix) of numbers used to filter what cells get passed to map algebra call-back function. 0 means a neighbor cell value should be treated as no-data and 1 means value should be treated as data. If weight is set to true, then the values, are used as multipliers to multiple the pixel value of that value in the neighborhood position.

weighted mask [x][x][x][x][x][x][x][x][x] ([x][x][x][x][x][x][x][x]) [x][x] (mask [x][x][x][x][x][x][x][x][x][x]
[x][x]) [x][x][x][x][x][x] ([x]/[x][x]) [x][x][x].

```
pixeltype pixeltype 0x000000, 0x00000000000000000000000000000000. pixeltype 0 NULL
0x00000000, 0x0000000000000000000000000000 (0x000000: INTERSECTION,
UNION, FIRST, CUSTOM), 0x0000000000000000 (0x000000: SECOND, LAST) 0x
0000000000000000. 0x0000000000000000. 0x00 pixeltype 0x00000000.
```

```
ST_BandPixelType ST_BandPixelType, ST_BandPixelType, NULL
```

extenttype `INTERSECTION()`, `UNION`, `FIRST(1)`, `SECOND`, `LAST`, `CUSTOM`.

```
customextent extenttype CUSTOM XXX, XXXX customextent XXXXXXXXXXXX. XX 1 X 4
XXXXXXXXXXXXXXXXXX.
```

distance_x The distance in pixels from the reference cell in x direction. So width of resulting matrix would be $2 \times \text{distance}_x + 1$. If not specified only the reference cell is considered (neighborhood of 0).

distancey $\times \times \times \times \times \times$ Y $\times \times \times \times \times \times \times \times \times \times \times \times$. $\times \times \times \times \times \times \times \times \times \times$ 2*distancey + 1 $\times \times \times \times$
 $\times$. $\times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times$ ($\times \times \times \times \times \times \times \times$ 0) $\times \times \times \times \times \times$.

```

userargs callbackfunc ██████████ variadic text ██████. ████████████████████
    █ callbackfunc ████████, userargs ██████████.

```


Note!

Note

(`VARIADIC`) `VARIADIC`, PostgreSQL `VARIADIC` `Query Language (SQL) Functions` "SQL Functions with Variable Numbers of Arguments".

Note!

Note

```
callbackfunc(text[]).
```

[illegible]

$\square\square 2 \square 3 \square\square\square\square\square\square 1 \square\square\square\square\square\square\square\square\square\square\square\square\square\square. \square\square 2 \square 3 \square\square\square\square\square\square\square\square\square\square.$

□□ 4 □□□□□□□□ 1 □□□□□□□□□□ 2 □□□□□□□□□□. □□ 4 □□□□□□□□□□.

2.2.0 0000 mask 000000000000.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.

实验: 实验 1

实验 1 实验, 实验 1 实验

```
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
```

实验 1 实验, 实验实验

```
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
```

实验实验, 实验实验

```
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
```

实验实验实验实验实验实验实验实验实验实验实验. 实验实验 PostgreSQL 9.1 实验实验实验实验实验实验.

```
WITH foo AS (
    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', 1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0) AS rast UNION ALL
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0) AS rast UNION ALL

    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10, 0) AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20, 0) AS rast UNION ALL
)
```

```

SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30, ←
0) AS rast UNION ALL

SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100, ←
0) AS rast UNION ALL
SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200, ←
0) AS rast UNION ALL
SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300, ←
0) AS rast
)
SELECT
  t1.rid,
  ST_MapAlgebra(
    ARRAY[ROW(ST_Union(t2.rast), 1)]::rastbandarg[],
    'sample_callbackfunc(double precision[], int[], text[])::regprocedure,
    '32BUI',
    'CUSTOM', t1.rast,
    1, 1
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 4
      AND t2.rid BETWEEN 0 AND 8
      AND ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rid, t1.rast

```

PostgreSQL 9.0 简体中文版。

```

WITH src AS (
  SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', ←
1, 0) AS rast UNION ALL
  SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0) ←
AS rast UNION ALL
  SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0) ←
AS rast UNION ALL

  SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10, ←
0) AS rast UNION ALL
  SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20, ←
0) AS rast UNION ALL
  SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30, ←
0) AS rast UNION ALL

  SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100, ←
0) AS rast UNION ALL
  SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200, ←
0) AS rast UNION ALL
  SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300, ←
0) AS rast
)
WITH foo AS (
  SELECT
    t1.rid,
    ST_Union(t2.rast) AS rast
  FROM src t1
  JOIN src t2
    ON ST_Intersects(t1.rast, t2.rast)
    AND t2.rid BETWEEN 0 AND 8
  WHERE t1.rid = 4
  GROUP BY t1.rid
), bar AS (
  SELECT

```

```

        t1.rid,
        ST_MapAlgebra(
            ARRAY[ROW(t2.rast, 1)::rastbandarg[],
                'raster_nmapalgebra_test(double precision[], int[], text[])'::regprocedure,
                '32BUI',
                'CUSTOM', t1.rast,
                1, 1
            ] AS rast
        FROM src t1
        JOIN foo t2
            ON t1.rid = t2.rid
    )
SELECT
    rid,
    (ST_Metadatas(rast)),
    (ST_BandMetadatas(rast, 1)),
    ST_Value(rast, 1, 1, 1)
FROM bar;

```

实验4-2

实验4-2-1, 实验4-2-2

```

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

```

实验4-2-1, 实验4-2-3

```

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

```

实验4-4

实验4-4-2, 实验4-4-3

```

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), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)

```

```

SELECT
  ST_MapAlgebra(
    t1.rast, 2,
    t2.rast, 1,
    'sample_callbackfunc(double precision[], int[], text[])::regprocedure'
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
      AND t2.rid = 2

```

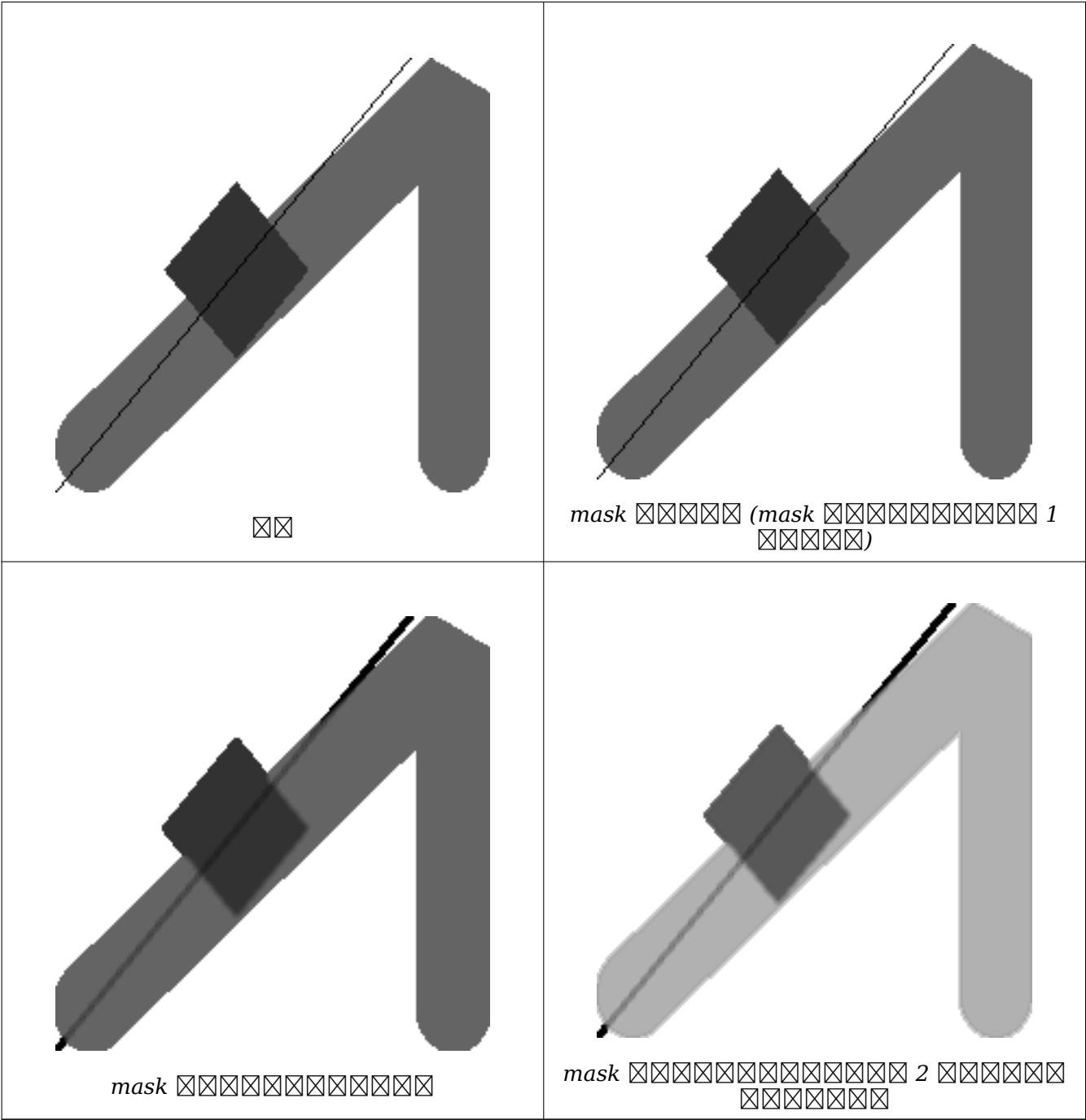
mask

```

WITH foo AS (SELECT
  ST_SetBandNoDataValue(
  ST_SetValue(ST_AsRaster(
    ST_Buffer(
      ST_GeomFromText('LINESTRING(50 50,100 90,100 50)'), 5,'join=bevel'),
      200,200,ARRAY['8BUI'], ARRAY[100], ARRAY[0]), ST_Buffer('POINT(70 70)::' ↵
        geometry,10,'quad_segs=1') ,50),
    'LINESTRING(20 20, 100 100, 150 98)::geometry,1),0) AS rast )
SELECT 'original' AS title, rast
FROM foo
UNION ALL
SELECT 'no mask mean value' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[], ↵
  int[], text[])::regprocedure) AS rast
FROM foo
UNION ALL
SELECT 'mask only consider neighbors, exclude center' AS title, ST_MapAlgebra(rast,1,' ↵
  ST_mean4ma(double precision[], int[], text[])::regprocedure,
  '{{1,1,1}, {1,0,1}, {1,1,1}}::double precision[], false) As rast
FROM foo

UNION ALL
SELECT 'mask weighted only consider neighbors, exclude center multi other pixel values by ↵
  2' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[], int[], text[]):: ↵
  regprocedure,
  '{{2,2,2}, {2,0,2}, {2,2,2}}::double precision[], true) As rast
FROM foo;

```



栅格

[rastbandarg](#), [ST_Union](#), [ST_MapAlgebra \(expression version\)](#)

11.12.6 ST_MapAlgebra (expression version)

ST_MapAlgebra (expression version) — 栅格 - 栅格 1 栅格 2 栅格, 栅格, 栅格
SQL 栅格 1 栅格 1 栅格.

Synopsis

raster **ST_MapAlgebra**(raster rast, integer nband, text pixeltype, text expression, double precision nodataval=NULL);
 raster **ST_MapAlgebra**(raster rast, text pixeltype, text expression, double precision nodataval=NULL);
 raster **ST_MapAlgebra**(raster rast1, integer nband1, raster rast2, integer nband2, text expression, text pixeltype=NULL, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
 raster **ST_MapAlgebra**(raster rast1, raster rast2, text expression, text pixeltype=NULL, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

返回

返回栅格 - 栅格 1 的 nband 2 个, 栅格 2, 栅格 1 和栅格 2 的 SQL 栅格 1 和栅格 2 的交集。

2.1.0 版本引入。

参数: 栅格 1, 2 (栅格 1 和 2)

栅格 1 (rast) 的 expression 表达式在 PostgreSQL 栅格函数中, 栅格 1 的 nband 和栅格 2 的 nband 相同, 栅格 1 的 pixeltype 和栅格 2 的 pixeltype 相同。

pixeltype 指定, 栅格 1 和栅格 2 的 pixeltype 相同。pixeltype 为 NULL 时, 栅格 1 和栅格 2 的 pixeltype 相同。

• expression 表达式。

1. [rast] - 栅格 1
2. [rast.val] - 栅格 1 的值
3. [rast.x] - 栅格 1 的 x 坐标
4. [rast.y] - 栅格 1 的 y 坐标

参数: 栅格 3, 4 (栅格 2 和 3)

栅格 1, (rast2) 的 expression 表达式在 PostgreSQL 栅格函数中, 栅格 1 的 nband1, band2 和栅格 2 的 nband1, band2 相同, 栅格 1 的 pixeltype 和栅格 2 的 pixeltype 相同。extenttype 指定, 栅格 1 和栅格 2 的 extenttype 相同。

expression 表达式在 PostgreSQL 栅格函数中, 栅格 1 和栅格 2 的 expression 表达式在 PostgreSQL 栅格函数中, 栅格 1 和栅格 2 的 expression 表达式在 PostgreSQL 栅格函数中。

pixeltype 指定, 栅格 1 和栅格 2 的 pixeltype 相同。ST_BandPixelType 函数在 PostgreSQL 栅格函数中, 栅格 1 和栅格 2 的 pixeltype 相同。

extenttype 指定

1. INTERSECTION - 栅格 1 和栅格 2 的交集。
2. UNION - 栅格 1 和栅格 2 的并集。
3. FIRST - 栅格 1 和栅格 2 的并集。

4. **SECOND** - 返回栅格中非空值的最小值。

nodataexpr rast1 非空值 NODATA 非空值 rast2 非空值
非空值, 返回 rast2 非空值。

nodata2expr rast2 非空值 NODATA 非空值 rast1 非空值
非空值, 返回 rast1 非空值。

nodatanodataval 非空值 rast1 非 rast2 非空值 NODATA 非空值
非空值。

• **expression, nodataexpr** 非 nodata2expr 非空值。

1. [rast1] - rast1 非空值
2. [rast1.val] - rast1 非空值
3. [rast1.x] - rast1 非空值 1-非空值
4. [rast1.y] - rast1 非空值 1-非空值
5. [rast2] - rast2 非空值
6. [rast2.val] - rast2 非空值
7. [rast2.x] - rast2 非空值 1-非空值
8. [rast2.y] - rast2 非空值 1-非空值

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, 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 — 对栅格 1 应用: 使用 PostgreSQL 表达式对栅格进行逐像素操作, 返回 1 个栅格。使用 1 个栅格, 返回 1 个栅格。

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 版本中, 使用 **ST_MapAlgebra (expression version)** 函数。

expression (rast) 对 **expression** 使用 PostgreSQL 表达式, 返回 1 个栅格。使用 1 个栅格, 返回 1 个栅格。使用 1 个栅格, 返回 1 个栅格。

pixeltype 指定, 使用 PostgreSQL 表达式。使用 NULL 指定, 使用 raster 指定。

使用 [rast], 1-指定 [rast.x], 1-指定 [rast.y] 指定。

2.0.0 版本。

返回

使用 2 个栅格 (modulo) 1 个栅格。

```
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

将 2BUI 的值设置为 0，如果 2BUI 的值在 100 和 250 之间，则设置为 1，如果 2BUI 的值在 252 和 254 之间，则设置为 2，否则设置为 0。

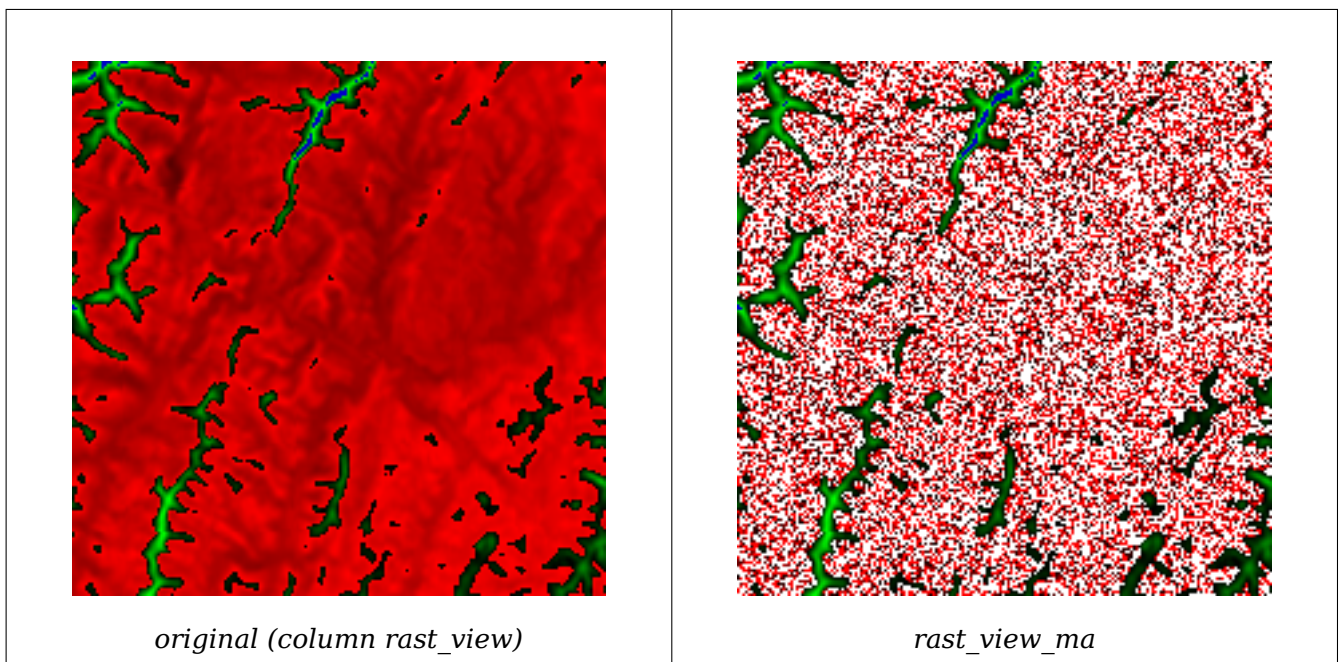
```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
UPDATE dummy_rast SET
  map_rast2 = ST_MapAlgebraExpr(rast, '2BUI'::text, 'CASE WHEN [rast] BETWEEN 100 and 250
    THEN 1 WHEN [rast] = 252 THEN 2 WHEN [rast] BETWEEN 253 and 254 THEN 3 ELSE 0 END'::
    text, '0')
WHERE rid = 2;
```

```
SELECT DISTINCT
  ST_Value(rast,1,i,j) As origval,
  ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 5) AS i
CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;
```

origval		mapval
249		1
250		1
251		1
252		2
253		3
254		3

```
SELECT
  ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast
WHERE rid = 2;
```

b1pixtyp
2BUI

[illegible]

```

SELECT
    ST_AddBand(
        ST_AddBand(
            ST_AddBand(
                ST_MakeEmptyRaster(rast_view),
                ST_MapAlgebraExpr(rast_view,1,NULL,'tan([rast])*[rast]')
            ),
            ST_Band(rast_view,2)
        ),
        ST_Band(rast_view, 3)
    ) As rast_view_ma
FROM wind
WHERE rid=167;

```



ST MapAlgebraExpr, ST MapAlgebraFct, ST BandPixelType, ST GeoReference, ST Value

11.12.8 ST_MapAlgebraExpr

ST_MapAlgebraExpr — 2 2: PostgreSQL 11.1, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0, 21.0, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 30.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0, 38.0, 39.0, 40.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 47.0, 48.0, 49.0, 50.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 60.0, 61.0, 62.0, 63.0, 64.0, 65.0, 66.0, 67.0, 68.0, 69.0, 70.0, 71.0, 72.0, 73.0, 74.0, 75.0, 76.0, 77.0, 78.0, 79.0, 80.0, 81.0, 82.0, 83.0, 84.0, 85.0, 86.0, 87.0, 88.0, 89.0, 90.0, 91.0, 92.0, 93.0, 94.0, 95.0, 96.0, 97.0, 98.0, 99.0, 100.0, 101.0, 102.0, 103.0, 104.0, 105.0, 106.0, 107.0, 108.0, 109.0, 110.0, 111.0, 112.0, 113.0, 114.0, 115.0, 116.0, 117.0, 118.0, 119.0, 120.0, 121.0, 122.0, 123.0, 124.0, 125.0, 126.0, 127.0, 128.0, 129.0, 130.0, 131.0, 132.0, 133.0, 134.0, 135.0, 136.0, 137.0, 138.0, 139.0, 140.0, 141.0, 142.0, 143.0, 144.0, 145.0, 146.0, 147.0, 148.0, 149.0, 150.0, 151.0, 152.0, 153.0, 154.0, 155.0, 156.0, 157.0, 158.0, 159.0, 160.0, 161.0, 162.0, 163.0, 164.0, 165.0, 166.0, 167.0, 168.0, 169.0, 170.0, 171.0, 172.0, 173.0, 174.0, 175.0, 176.0, 177.0, 178.0, 179.0, 180.0, 181.0, 182.0, 183.0, 184.0, 185.0, 186.0, 187.0, 188.0, 189.0, 190.0, 191.0, 192.0, 193.0, 194.0, 195.0, 196.0, 197.0, 198.0, 199.0, 200.0, 201.0, 202.0, 203.0, 204.0, 205.0, 206.0, 207.0, 208.0, 209.0, 210.0, 211.0, 212.0, 213.0, 214.0, 215.0, 216.0, 217.0, 218.0, 219.0, 220.0, 221.0, 222.0, 223.0, 224.0, 225.0, 226.0, 227.0, 228.0, 229.0, 230.0, 231.0, 232.0, 233.0, 234.0, 235.0, 236.0, 237.0, 238.0, 239.0, 240.0, 241.0, 242.0, 243.0, 244.0, 245.0, 246.0, 247.0, 248.0, 249.0, 250.0, 251.0, 252.0, 253.0, 254.0, 255.0, 256.0, 257.0, 258.0, 259.0, 260.0, 261.0, 262.0, 263.0, 264.0, 265.0, 266.0, 267.0, 268.0, 269.0, 270.0, 271.0, 272.0, 273.0, 274.0, 275.0, 276.0, 277.0, 278.0, 279.0, 280.0, 281.0, 282.0, 283.0, 284.0, 285.0, 286.0, 287.0, 288.0, 289.0, 290.0, 291.0, 292.0, 293.0, 294.0, 295.0, 296.0, 297.0, 298.0, 299.0, 300.0, 301.0, 302.0, 303.0, 304.0, 305.0, 306.0, 307.0, 308.0, 309.0, 310.0, 311.0, 312.0, 313.0, 314.0, 315.0, 316.0, 317.0, 318.0, 319.0, 320.0, 321.0, 322.0, 323.0, 324.0, 325.0, 326.0, 327.0, 328.0, 329.0, 330.0, 331.0, 332.0, 333.0, 334.0, 335.0, 336.0, 337.0, 338.0, 339.0, 340.0, 341.0, 342.0, 343.0, 344.0, 345.0, 346.0, 347.0, 348.0, 349.0, 350.0, 351.0, 352.0, 353.0, 354.0, 355.0, 356.0, 357.0, 358.0, 359.0, 360.0, 361.0, 362.0, 363.0, 364.0, 365.0, 366.0, 367.0, 368.0, 369.0, 370.0, 371.0, 372.0, 373.0, 374.0, 375.0, 376.0, 377.0, 378.0, 379.0, 380.0, 381.0, 382.0, 383.0, 384.0, 385.0, 386.0, 387.0, 388.0, 389.0, 390.0, 391.0, 392.0, 393.0, 394.0, 395.0, 396.0, 397.0, 398.0, 399.0, 400.0, 401.0, 402.0, 403.0, 404.0, 405.0, 406.0, 407.0, 408.0, 409.0, 410.0, 411.0, 412.0, 413.0, 414.0, 415.0, 416.0, 417.0, 418.0, 419.0, 420.0, 421.0, 422.0, 423.0, 424.0, 425.0, 426.0, 427.0, 428.0, 429.0, 430.0, 431.0, 432.0, 433.0, 434.0, 435.0, 436.0, 437.0, 438.0, 439.0, 440.0, 441.0, 442.0, 443.0, 444.0, 445.0, 446.0, 447.0, 448.0, 449.0, 450.0, 451.0, 452.0, 453.0, 454.0, 455.0, 456.0, 457.0, 458.0, 459.0, 460.0, 461.0, 462.0, 463.0, 464.0, 465.0, 466.0, 467.0, 468.0, 469.0, 470.0, 471.0, 472.0, 473.0, 474.0, 475.0, 476.0, 477.0, 478.0, 479.0, 480.0, 481.0, 482.0, 483.0, 484.0, 485.0, 486.0, 487.0, 488.0, 489.0, 490.0, 491.0, 492.0, 493.0, 494.0, 495.0, 496.0, 497.0, 498.0, 499.0, 500.0, 501.0, 502.0, 503.0, 504.0, 505.0, 506.0, 507.0, 508.0, 509.0, 510.0, 511.0, 512.0, 513.0, 514.0, 515.0, 516.0, 517.0, 518.0, 519.0, 520.0, 521.0, 522.0, 523.0, 524.0, 525.0, 526.0, 527.0, 528.0, 529.0, 530.0, 531.0, 532.0, 533.0, 534.0, 535.0, 536.0, 537.0, 538.0, 539.0, 540.0, 541.0, 542.0, 543.0, 544.0, 545.0, 546.0, 547.0, 548.0, 549.0, 550.0, 551.0, 552.0, 553.0, 554.0, 555.0, 556.0, 557.0, 558.0, 559.0, 560.0, 561.0, 562.0, 563.0, 564.0, 565.0, 566.0, 567.0, 568.0, 569.0, 570.0, 571.0, 572.0, 573.0, 574.0, 575.0, 576.0, 577.0, 578.0, 579.0, 580.0, 581.0, 582.0, 583.0, 584.0, 585.0, 586.0, 587.0, 588.0, 589.0, 590.0, 591.0, 592.0, 593.0, 594.0, 595.0, 596.0, 597.0, 598.0, 599.0, 600.0, 601.0, 602.0, 603.0, 604.0, 605.0, 606.0

Synopsis

```

raster ST_MapAlgebraExpr(raster rast1, raster rast2, text expression, text pixelttype=same_as_rast1_band
text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double preci-

```

sion nodatanodataval=NULL);
 raster **ST_MapAlgebraExpr**(raster rast1, integer band1, raster rast2, integer band2, text expression, text pixeltype=same_as_rast1_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

警告



Warning

ST_MapAlgebraExpr 2.1.0 版本中，**ST_MapAlgebra (expression version)** 函数已被弃用。

rast1, (**rast2**) 是 **expression** 中出现的 2 个 raster，都是 PostgreSQL 支持的 raster 类型，且 1 个是 32 位，另一个是 64 位。 **band1**, **band2** 是 1 到 255 之间的整数，表示要使用的 band。 **expression** 是 PostgreSQL 支持的表达式，可以是 **ST_MapAlgebraExpr** 函数本身。 **pixeltype** 是 PostgreSQL 支持的 raster 类型。

expression 是 2 个 raster 的 PostgreSQL 表达式，可以是 **ST_MapAlgebraExpr** 函数本身。 **PostgreSQL** 支持的表达式。 **band1**, **band2** 是 1 到 255 之间的整数，表示要使用的 band。

pixeltype 是 PostgreSQL 支持的 raster 类型。 **ST_BandPixelType** 函数返回 raster 的 pixeltype，可以是 NULL，表示 raster 的 pixeltype 是 NULL。

extenttype 是 PostgreSQL 支持的 raster 类型。

1. INTERSECTION - 返回两个 raster 的交集。
2. UNION - 返回两个 raster 的并集。
3. FIRST - 返回两个 raster 中的第一个。
4. SECOND - 返回两个 raster 中的第二个。

nodata1expr rast1 是 NODATA 的表达式，rast2 是 NODATA 的表达式。

nodata2expr rast2 是 NODATA 的表达式，rast1 是 NODATA 的表达式。

nodatanodataval 是 rast1 和 rast2 的 NODATA 值。

pixeltype 是 PostgreSQL 支持的 raster 类型。 **pixeltype** 是 NULL，表示 raster 的 pixeltype 是 NULL。

extenttype 是 PostgreSQL 支持的 raster 类型。 **extenttype** 是 NULL，表示 raster 的 extenttype 是 NULL。

2.0.0 版本中，**ST_MapAlgebraExpr** 函数已被弃用。

警告： 2 个 raster

2 个 raster (modulo) 1 个 raster。

```
--Create a cool set of rasters --
DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

-- Insert some cool shapes around Boston in Massachusetts state plane meters --
INSERT INTO fun_shapes(fun_name, rast)
VALUES ('ref', ST_AsRaster(ST_MakeEnvelope(235229, 899970, 237229, 901930,26986),200,200,'8BUI',0,0));

INSERT INTO fun_shapes(fun_name,rast)
WITH ref(rast) AS (SELECT rast FROM fun_shapes WHERE fun_name = 'ref' )
SELECT 'area' AS fun_name, ST_AsRaster(ST_Buffer(ST_SetSRID(ST_Point(236229, 900930),26986)
, 1000),
    ref.rast,'8BUI', 10, 0) As rast
FROM ref
UNION ALL
SELECT 'rand bubbles',
    ST_AsRaster(
        (SELECT ST_Collect(geom)
        FROM (SELECT ST_Buffer(ST_SetSRID(ST_Point(236229 + i*random()*100, 900930 + j*random()
*100),26986), random()*20) As geom
        FROM generate_series(1,10) As i, generate_series(1,10) As j
        ) As foo ), ref.rast,'8BUI', 200, 0)
FROM ref;

--map them -
SELECT ST_MapAlgebraExpr(
    area.rast, bub.rast, '[rast2.val]', '8BUI', 'INTERSECTION', '[rast2.val]', '[rast1.
val]') As interrast,
    ST_MapAlgebraExpr(
    area.rast, bub.rast, '[rast2.val]', '8BUI', 'UNION', '[rast2.val]', '[rast1.val
]') As unionrast
FROM
    (SELECT rast FROM fun_shapes WHERE
    fun_name = 'area') As area
CROSS JOIN (SELECT rast
FROM fun_shapes WHERE
    fun_name = 'rand bubbles') As bub
```

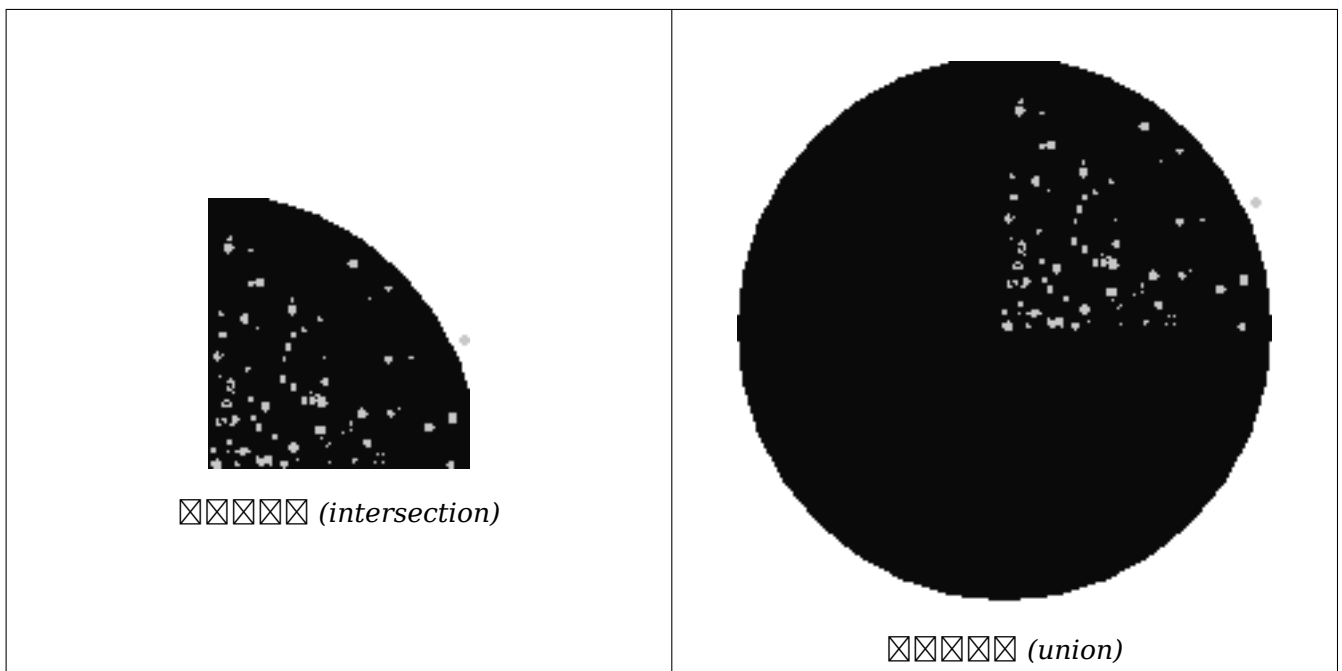


图 10-10: 两个圆的交集和并集

```
-- 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;
```

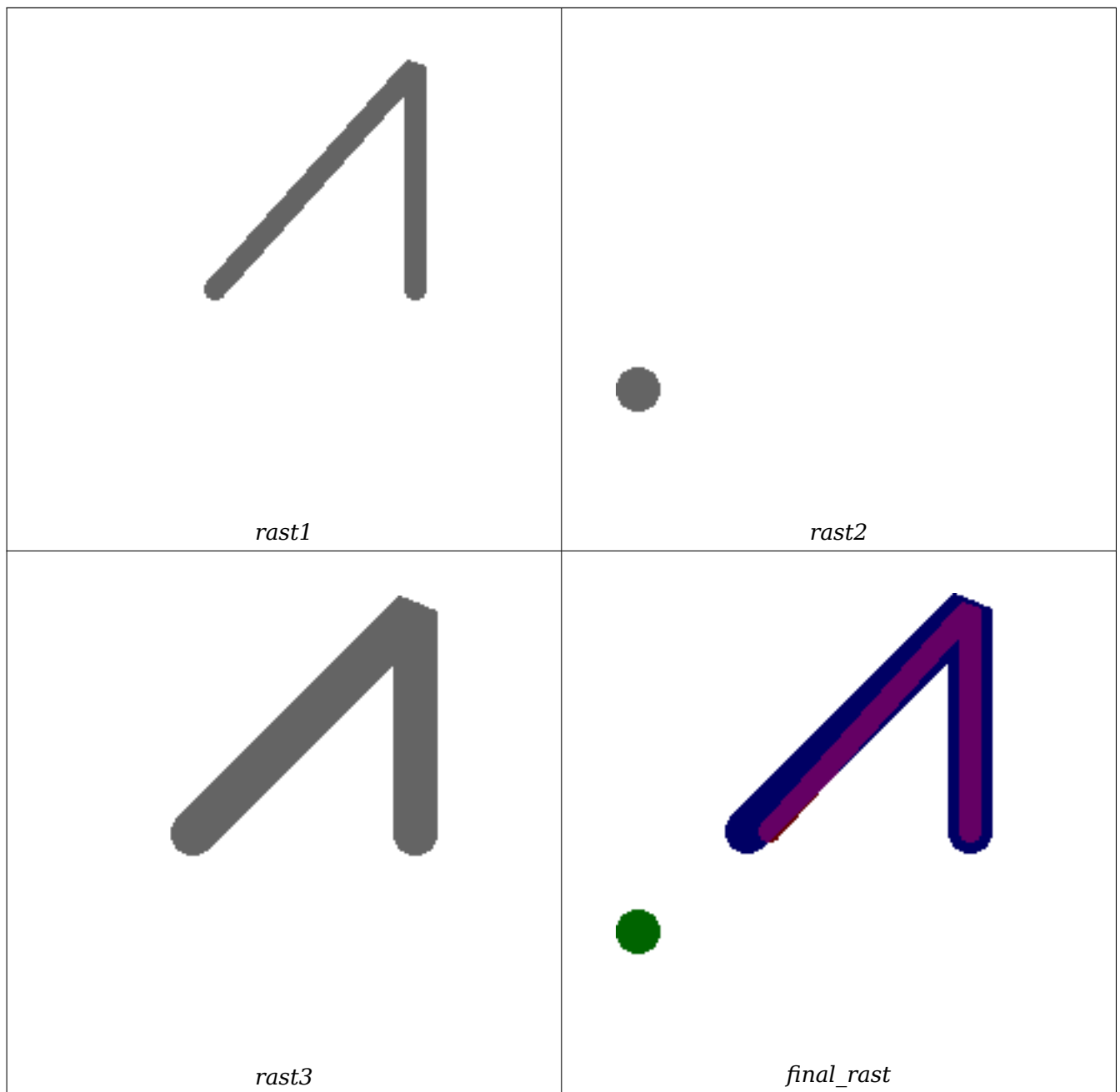


图 10-10: 创建 3 波段栅格

```
-- Create new 3 band raster composed of first 2 clipped bands, and overlay of 3rd band with ←
  our geometry
-- This query took 3.6 seconds on PostGIS windows 64-bit install
WITH pr AS
-- Note the order of operation: we clip all the rasters to dimensions of our region
(SELECT ST_Clip(rast,ST_Expand(geom,50) ) As rast, g.geom
 FROM aerals.o_2_boston AS r INNER JOIN
-- union our parcels of interest so they form a single geometry we can later intersect with
  (SELECT ST_Union(ST_Transform(geom,26986)) AS geom
   FROM landparcels WHERE pid IN('0303890000', '0303900000')) As g
   ON ST_Intersects(rast::geometry, ST_Expand(g.geom,50))
),
```

```
-- we then union the raster shards together
-- ST_Union on raster is kinda of slow but much faster the smaller you can get the rasters
-- therefore we want to clip first and then union
prunion AS
(SELECT ST_AddBand(NULL, ARRAY[ST_Union(rast,1),ST_Union(rast,2),ST_Union(rast,3)] ) As ←
    clipped,geom
FROM pr
GROUP BY geom)
-- return our final raster which is the unioned shard with
-- with the overlay of our parcel boundaries
-- add first 2 bands, then mapalgebra of 3rd band + geometry
SELECT ST_AddBand(ST_Band(clipped,ARRAY[1,2])
    , ST_MapAlgebraExpr(ST_Band(clipped,3), ST_AsRaster(ST_Buffer(ST_Boundary(geom),2), ←
        clipped, '8BUI',250),
        '[rast2.val]', '8BUI', 'FIRST', '[rast2.val]', '[rast1.val]')) ) As rast
FROM prunion;
```



图 11.12.9: 使用 ST_MapAlgebraFct 函数处理栅格数据。

图 11.12.9

[ST_MapAlgebraExpr](#), [ST_AddBand](#), [ST_AsPNG](#), [ST_AsRaster](#), [ST_MapAlgebraFct](#), [ST_BandPixelType](#), [ST_GeoReference](#), [ST_Value](#), [ST_Union](#), [ST_Union](#)

11.12.9 ST_MapAlgebraFct

ST_MapAlgebraFct — 对栅格数据应用 1 个函数: 对 PostgreSQL 栅格数据应用 1 个函数, 对 1 个栅格数据应用 1 个函数。对 1 个栅格数据应用 1 个函数。

Synopsis

```

raster ST_MapAlgebraFct(raster rast, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster ST_MapAlgebraFct(raster rast, text pixeltype, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster ST_MapAlgebraFct(raster rast, integer band, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, integer band, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster ST_MapAlgebraFct(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);

```

注意



Warning

ST_MapAlgebraFct 2.1.0 版本中，**ST_MapAlgebra** (callback function version) 已被弃用。

rast (rast) 是 **onerasteruserfunc** 在 PostgreSQL 中注册的函数名，**band** 是 1 到 **rast** 的带号。如果 **band** 为 1，则使用所有带。如果 **band** 为 0，则使用所有带。

pixeltype 是像素的数据类型。如果 **pixeltype** 为 NULL，则使用 **rast** 的像素数据类型。

The **onerasteruserfunc** parameter must be the name and signature of a SQL or PL/pgSQL function, cast to a regprocedure. A very simple and quite useless PL/pgSQL function example is:

```

CREATE OR REPLACE FUNCTION simple_function(pixel FLOAT, pos INTEGER[], VARIADIC args TEXT [])
RETURNS FLOAT
AS $$ BEGIN
    RETURN 0.0;
END; $$
LANGUAGE 'plpgsql' IMMUTABLE;

```

The userfunction may accept two or three arguments: a float value, an optional integer array, and a variadic text array. The first argument is the value of an individual raster cell (regardless of the raster datatype). The second argument is the position of the current processing cell in the form '{x,y}'. The third argument indicates that all remaining parameters to **ST_MapAlgebraFct** shall be passed through to the userfunction.

Passing a regprodedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'simple_function(float,integer[],text[])::regprocedure'
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

userfunction 是用户函数的名称，**args** 是传递给 **ST_MapAlgebraFct** 的变长文本数组。

**Note**

(`ST_MapAlgebraFct`) VARIADIC 函数在 PostgreSQL 的 [Query Language \(SQL\) Functions](#) 中“SQL Functions with Variable Numbers of Arguments”章节中有描述。

**Note**

`ST_MapAlgebraFct` 函数支持 `userfunction` 文本数组 `text[]` 参数。

2.0.0 版本中的示例。

SQL

创建 2 个栅格 (modulo) 函数，将 1 个栅格转换为 2 个栅格。

```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
CREATE FUNCTION mod_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS $$
BEGIN
    RETURN pixel::integer % 2;
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE;

UPDATE dummy_rast SET map_rast = ST_MapAlgebraFct(rast,NULL,'mod_fct(float,integer[],text ↵
[]')::regprocedure) WHERE rid = 2;

SELECT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

origval	mapval
253	1
254	0
253	1
253	1
254	0
254	0
254	0
250	0
254	0
254	0

创建 NODATA 函数 (0) 将 2BUI 栅格转换为 1 个栅格。

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
CREATE FUNCTION classify_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
DECLARE
    nodata float := 0;
BEGIN
    IF NOT args[1] IS NULL THEN
```

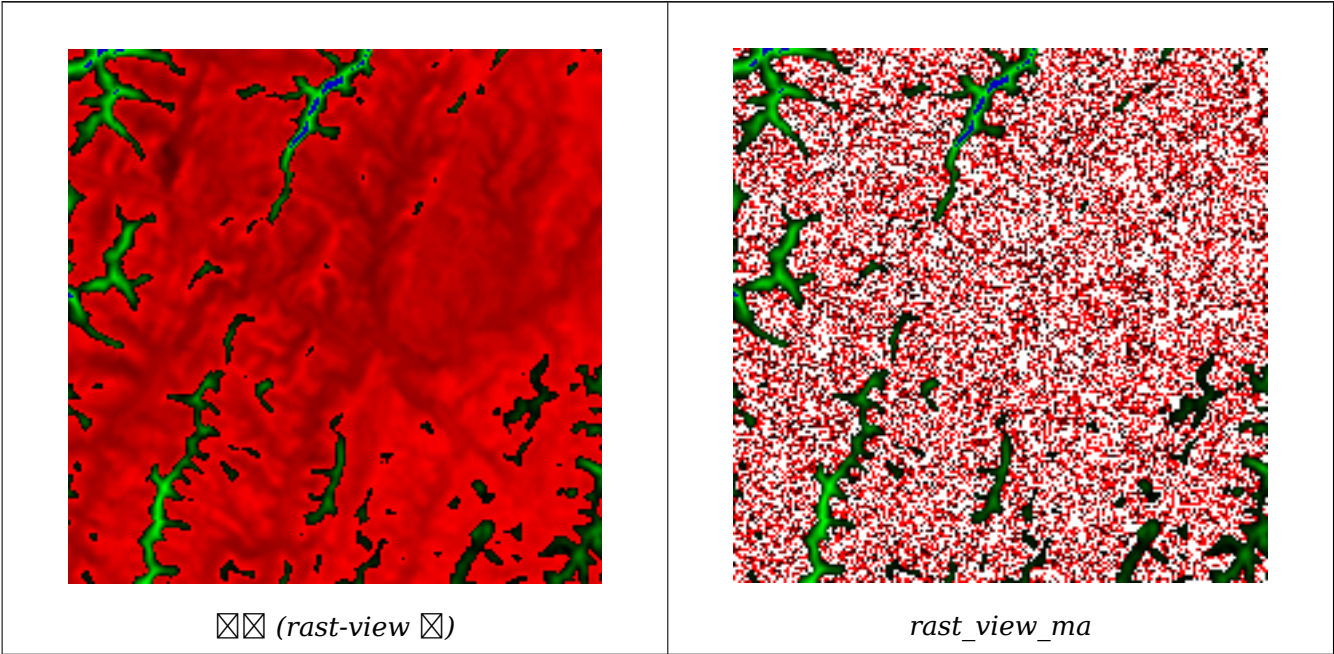
```
        noata := args[1];
    END IF;
    IF pixel < 251 THEN
        RETURN 1;
    ELSIF pixel = 252 THEN
        RETURN 2;
    ELSIF pixel
> 252 THEN
        RETURN 3;
    ELSE
        RETURN noata;
    END IF;
END;
$$
LANGUAGE 'plpgsql';
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ↵
[],text[])'::regprocedure, '0') WHERE rid = 2;

SELECT DISTINCT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 5) AS i CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;

origval | mapval
-----+-----
    249 |      1
    250 |      1
    251 |
    252 |      2
    253 |      3
    254 |      3

SELECT ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast WHERE rid = 2;

b1pixtyp
-----
2BUI
```



`rast1`, `rast2` 是 `tworastuserfunc` 在 PostgreSQL 中定义的函数，第 1 个参数是 `rast1`，第 2 个参数是 `rast2`，第 3 个参数是 `band1`，第 4 个参数是 `band2`，第 5 个参数是 `pos`。函数返回一个栅格值。

`pixeltype` 是栅格的像素数据类型。如果 `pixeltype` 是 NULL，则返回 `rast1` 的像素数据类型。

The `tworastuserfunc` parameter must be the name and signature of an SQL or PL/pgSQL function, cast to a regprocedure. An example PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION simple_function_for_two_rasters(pixel1 FLOAT, pixel2 FLOAT, pos
  INTEGER[], VARIADIC args TEXT[])
  RETURNS FLOAT
  AS $$ BEGIN
    RETURN 0.0;
  END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

The `tworastuserfunc` may accept three or four arguments: a double precision value, a double precision value, an optional integer array, and a variadic text array. The first argument is the value of an individual raster cell in `rast1` (regardless of the raster datatype). The second argument is an individual raster cell value in `rast2`. The third argument is the position of the current processing cell in the form '{x,y}'. The fourth argument indicates that all remaining parameters to `ST_MapAlgebraFct` shall be passed through to the `tworastuserfunc`.

Passing a regprocedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'simple_function(double precision, double precision, integer[], text[])::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

The fourth argument to the `tworastuserfunc` is a variadic text array. All trailing text arguments to any `ST_MapAlgebraFct` call are passed through to the specified `tworastuserfunc`, and are contained in the `userargs` argument.



Note

(`VARIADIC`) 是 PostgreSQL 中的一个关键字，用于定义变长参数。有关更多信息，请参阅 [Query Language \(SQL\) Functions](#) 中的“SQL Functions with Variable Numbers of Arguments”部分。



Note

在 `tworastuserfunc` 中，`text[]` 是一个变长文本数组。

2.0.0 版本中已弃用。

示例：

```
-- define our user defined function --
CREATE OR REPLACE FUNCTION raster_mapalgebra_union(
  rast1 double precision,
```

```

    rast2 double precision,
    pos integer[],
    VARIADIC userargs text[]
)
RETURNS double precision
AS $$
DECLARE
BEGIN
    CASE
        WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
            RETURN ((rast1 + rast2)/2.);
        WHEN rast1 IS NULL AND rast2 IS NULL THEN
            RETURN NULL;
        WHEN rast1 IS NULL THEN
            RETURN rast2;
        ELSE
            RETURN rast1;
    END CASE;

    RETURN NULL;
END;
$$ LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- prep our test table of rasters
DROP TABLE IF EXISTS map_shapes;
CREATE TABLE map_shapes(rid serial PRIMARY KEY, rast raster, bnum integer, descrip text);
INSERT INTO map_shapes(rast,bnum, descrip)
WITH mygeoms
  AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(90,90),30) As geom, 'circle' As descrip
        UNION ALL
        SELECT 3 AS bnum,
            ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 15) As geom, ←
            'big road' As descrip
        UNION ALL
        SELECT 1 As bnum,
            ST_Translate(ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'), ←
            8,'join=bevel'), 10,-6) As geom, 'small road' As descrip
    ),
-- define our canvas to be 1 to 1 pixel to geometry
canvas
  AS ( SELECT ST_AddBand(ST_MakeEmptyRaster(250,
    250,
    ST_XMin(e)::integer, ST_YMax(e)::integer, 1, -1, 0, 0 ) , '8BUI'::text,0) As rast
    FROM (SELECT ST_Extent(geom) As e,
            Max(ST_SRID(geom)) As srid
          from mygeoms
        ) As foo
    )
-- return our rasters aligned with our canvas
SELECT ST_AsRaster(m.geom, canvas.rast, '8BUI', 240) As rast, bnum, descrip
  FROM mygeoms AS m CROSS JOIN canvas
UNION ALL
SELECT canvas.rast, 4, 'canvas'
FROM canvas;

-- Map algebra on single band rasters and then collect with ST_AddBand
INSERT INTO map_shapes(rast,bnum,descrip)
SELECT ST_AddBand(ST_AddBand(rasts[1], rasts[2]),rasts[3]), 4, 'map bands overlay fct union ←
  (canvas)'
FROM (SELECT ARRAY(SELECT ST_MapAlgebraFct(m1.rast, m2.rast,
  'raster_mapalgebra_union(double precision, double precision, integer[], text[]) ←
  '::regprocedure, '8BUI', 'FIRST')

```

```
FROM map_shapes As m1 CROSS JOIN map_shapes As m2
WHERE m1.descrip = 'canvas' AND m2.descrip <
> 'canvas' ORDER BY m2.bnum) As rasts) As foo;
```



图 4-1-1 (R: small road, G: circle, B: big road)

实验四 栅格数据操作

```
CREATE OR REPLACE FUNCTION raster_mapalgebra_userargs(
  rast1 double precision,
  rast2 double precision,
  pos integer[],
  VARIADIC userargs text[]
)
RETURNS double precision
AS $$
DECLARE
BEGIN
  CASE
    WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
      RETURN least(userargs[1]::integer,(rast1 + rast2)/2.);
    WHEN rast1 IS NULL AND rast2 IS NULL THEN
      RETURN userargs[2]::integer;
    WHEN rast1 IS NULL THEN
      RETURN greatest(rast2,random()*userargs[3]::integer)::integer;
    ELSE
      RETURN greatest(rast1, random()*userargs[4]::integer)::integer;
  END CASE;

  RETURN NULL;
END;
$$ LANGUAGE 'plpgsql' VOLATILE COST 1000;
```

```
SELECT ST_MapAlgebraFct(m1.rast, 1, m1.rast, 3,
    'raster_mapalgebra_userargs(double precision, double precision, integer[], text ←
    [])'::regprocedure,
    '8BUI', 'INTERSECT', '100','200','200','0')
FROM map_shapes As m1
WHERE m1.descrip = 'map bands overlay fct union (canvas)';
```

[illegible]

ST_MapAlgebraExpr, ST_BandPixelType, ST_GeoReference, ST_SetValue

11.12.11 ST_MapAlgebraFctNgb

ST_MapAlgebraFctNgb — 1 : PostgreSQL
 (Map Algebra Nearest Neighbor) . (neighborhood) PostgreSQL
.

Synopsis

```
raster ST_MapAlgebraFctNgb(raster rast, integer band, text pixelttype, integer ngbwidth, integer  
ngbheight, regprocedure onerastngbuserfunc, text nodatamode, text[] VARIADIC args);
```



Warning

Warning
ST_MapAlgebraFctNgb 2.1.0 . ST_MapAlgebra
(callback function version) .

Figure 1: Neighborhood PostgreSQL database schema. Neighborhoods, users, and items are represented as tables. The schema includes tables for neighborhoods, users, items, and their relationships, with various attributes and foreign key constraints.

rast

band ☐☐☐☐☐☐☐☐☐☐ (☐☐☐☐ 1)

pixeltype 000000000000. 0000 ST_BandPixelType 00000000000000000000,
00000, NULL 0000000000. 0000000000 NULL 0000000000, rast 000000
000000000000. 0000000000000000000000000000000000000000000000000000.

ngbwidth ☒☒ (neighborhood) ☒☒☒☒☒☒

ngbheight ☒☒ (neighborhood) ☒☒☒☒☒☒

[illegible]

nodatamode NODATA ☐☐ NULL ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐.

```
'ignore': 0x0000000000000000 NODATA 0x0000000000000000. 0x00000000000000000000000000000000
0x0000000000000000 NODATA 0x00000000000000000000000000000000.
```

```
'NULL': 00000000 NODATA 00000000 NULL 00000000. 000000000000
00000000.
```

[illegible]

args ☒☒☒☒☒☒☒☒☒☒☒☒

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.



ST Rescale <https://gdal.org/user/drivers/raster/postgisraster.html>.

```
-- A simple 'callback' user function that averages up all the values in a neighborhood.
--
CREATE OR REPLACE FUNCTION rast_avg(matrix float[][], nodatamode text, variadic args text ↔
[])
RETURNS float AS
$$
DECLARE
    _matrix float[][];
    x1 integer;
    x2 integer;
    y1 integer;
    y2 integer;
    sum float;
BEGIN
    _matrix := matrix;
    sum := 0;
    FOR x in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
        FOR y in array_lower(matrix, 2)..array_upper(matrix, 2) LOOP
            sum := sum + _matrix[x][y];
        END LOOP;
    END LOOP;
    RETURN (sum*1.0/(array_upper(matrix,1)*array_upper(matrix,2) ))::integer ;
END;
```

```

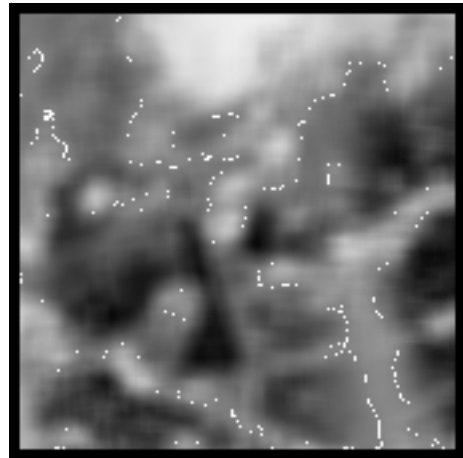
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- now we apply to our raster averaging pixels within 2 pixels of each other in X and Y ↔
direction --
SELECT ST_MapAlgebraFctNgb(rast, 1, '8BUI', 4,4,
    'rast_avg(float[][], text, text[])::regprocedure, 'NULL', NULL) As nn_with_border
FROM katrinas_rescaled
limit 1;

```



输入栅格



应用 4x4 邻域平均函数后的栅格

☐☐

[ST_MapAlgebraFct](#), [ST_MapAlgebraExpr](#), [ST_Rescale](#)

11.12.12 ST_Reclass

ST_Reclass — 对输入栅格中的每个像素应用重新分类表达式。nband 指定要处理的波段数。nband 为 1 时，默认处理所有波段。像素类型指定输出栅格的像素类型。16BUI 指定 8BUI 的输出精度。

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 安装与使用.

Example: Basic

创建 2 个 8BUI 的 4BUI 栅格，101 个 254 个 NODATA 栅格。

```
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

创建 1, 2, 3 个 1BB, 4BUI, 4BUI 栅格。

创建 (1, 2, 3) 个 1BB, 4BUI, 4BUI 栅格。 (1, 2, 3) 个 1BB, 4BUI, 4BUI 栅格。 reclassarg 栅格。

```
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 数据类型

32BF 数据类型与 32 位的 IEEE 754 浮点格式兼容。3 个 ((8BUI,8BUI,8BUI) 数据类型。

```
ALTER TABLE wind ADD COLUMN rast_view raster;
UPDATE wind
  set rast_view = ST_AddBand( NULL,
    ARRAY[
      ST_Reclass(rast, 1, '0.1-10]:1-10,9-10]:11,(11-33:0':::text, '8BUI':::text,0),
      ST_Reclass(rast,1, '11-33):0-255,[0-32:0,(34-1000:0':::text, '8BUI':::text,0),
      ST_Reclass(rast,1,'0-32]:0,(32-100:100-255':::text, '8BUI':::text,0)
    ]
  );
```

☐☐

[ST_AddBand](#), [ST_Band](#), [ST_BandPixelType](#), [ST_MakeEmptyRaster](#), [reclassarg](#), [ST_Value](#)

11.12.13 ST_ReclassExact

ST_ReclassExact — Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.

Synopsis

raster **ST_ReclassExact**(raster rast, double precision[] inputvalues, double precision[] outputvalues, integer bandnumber=1, text pixeltype=32BF, double precision nodatavalue=NULL);

☐☐

Creates a new raster formed by applying a reclassification operation defined by the `inputvalues` and `outputvalues` arrays. Pixel values found in the input array are mapped to the corresponding value in the output array. All other pixel values are mapped to the `nodatavalue`.

The output pixel type defaults to float, but can be specified using the `pixeltype` parameter. If no `bandnumber` is specified band 1 is assumed.

The new raster will have the same georeference, width, and height as the original raster. Bands not designated are returned unchanged.

Availability: 3.6.0

Example: Basic

Create a small raster and map its pixels to new values.

```
CREATE TABLE reclassexact (
    id integer,
    rast raster
);

--
-- Create a raster with just four pixels
-- [1 2]
-- [3 4]
--
INSERT INTO reclassexact (id, rast)
SELECT 1, ST_SetValues(
    ST_AddBand(
        ST_MakeEmptyRaster(
            2,      -- width in pixels
            2,      -- height in pixels
            0,      -- upper-left x-coordinate
            0,      -- upper-left y-coordinate
            1,      -- pixel size in x-direction
            -1,     -- pixel size in y-direction (negative for north-up)
            0,      -- skew in x-direction
            0,      -- skew in y-direction
            4326    -- SRID (e.g., WGS 84)
        ),
        '32BUI'::text, -- pixel type (e.g., '32BF' for float, '8BUI' for unsigned 8-bit int)
        0.0,          -- initial value for the band (e.g., 0.0 or a no-data value)
        -99           -- nodatavalue
    ),
    1, -- band number (usually 1 for single-band rasters)
    1, -- x origin for setting values (usually 1)
    1, -- y origin for setting values (usually 1)
    ARRAY[
        ARRAY[1, 2],
        ARRAY[3, 4]
    ]::double precision[][] -- 2D array of values
);

-- Reclass the values to new values
-- and dump the values of the new raster for display
WITH rc AS (
    SELECT ST_ReclassExact(
        rast,          -- input raster
        ARRAY[4,3,2,1], -- input map
        ARRAY[14,13,12,11], -- output map
        1,             -- band number to remap
        '32BUI'        -- output raster pixeltype
    ) AS rast
    FROM reclassexact
    WHERE id = 1
)
SELECT 'rce-1', (ST_DumpValues(rc.rast)).*
FROM rc;
```

☒☒

ST_Reclass, ST_AddBand, ST_Band, ST_MakeEmptyRaster

11.12.14 ST_Union

`ST_Union` — 将 1 个或多个栅格合并为一个栅格。

Synopsis

```
raster ST_Union(setof raster rast);
raster ST_Union(setof raster rast, unionarg[] unionargset);
raster ST_Union(setof raster rast, integer nband);
raster ST_Union(setof raster rast, text uniontype);
raster ST_Union(setof raster rast, integer nband, text uniontype);
```

返回

返回 1 个栅格。如果 `unionarg` 或 `uniontype` 被指定，则返回的栅格将具有指定的 `uniontype`。如果 `uniontype` 未指定，则返回的栅格将具有 `LAST(unionarg)`、`FIRST`、`MIN`、`MAX`、`COUNT`、`SUM`、`MEAN`、`RANGE` 中的第一个值。



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 (通过 C 函数实现)。

2.1.0 版本引入 `ST_Union(rast, unionarg)` 函数。

更新: 2.1.0 版本引入 `ST_Union(rast)` 函数 1 个参数。PostGIS 3.0 版本引入。

更新: 2.1.0 版本引入 `ST_Union(rast, uniontype)` 函数 4 个参数。

示例: 将栅格合并为一个栅格

```
-- this creates a single band from first band of raster tiles
-- that form the original file system tile
SELECT filename, ST_Union(rast,1) As file_rast
FROM sometable WHERE filename IN('dem01','dem02') GROUP BY filename;
```

示例: 将栅格合并为一个栅格

```
-- this creates a multi band raster collecting all the tiles that intersect a line
-- Note: In 2.0, this would have just returned a single band raster
-- , new union works on all bands by default
-- this is equivalent to unionarg: ARRAY[ROW(1, 'LAST'), ROW(2, 'LAST'), ROW(3, 'LAST')]:: ↵
unionarg[]
SELECT ST_Union(rast)
FROM aerials.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ↵
);
```

注意: 此函数返回的栅格数据是多带的。

以下 SQL 语句创建了一个多带栅格，收集了所有与一条线相交的瓦片。

```
-- this creates a multi band raster collecting all the tiles that intersect a line
SELECT ST_Union(rast,ARRAY[ROW(2, 'LAST'), ROW(1, 'LAST'), ROW(3, 'LAST')]::unionarg[])
FROM aerials.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

返回

`unionarg`, `ST_Envelope`, `ST_ConvexHull`, `ST_Clip`, `ST_Union`

11.13 栅格函数

11.13.1 ST_Distinct4ma

`ST_Distinct4ma` — 对 4 个相邻的栅格瓦片进行去重。

Synopsis

`float8 ST_Distinct4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);`
`double precision ST_Distinct4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);`

返回

返回一个与输入栅格具有相同尺寸和 SRID 的栅格。



Note

参数 1 是 `ST_MapAlgebraFctNgb` 函数。



Note

参数 2 是 `ST_MapAlgebra (callback function version)` 函数。



Warning

2.1.0 版本中 `ST_MapAlgebraFctNgb` 函数参数 1 是必需的。

2.0.0 版本中。

返回: 2.1.0 版本中 2 个参数。

例

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_distinct4ma(float[],text,text[])':: ↵
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |      3
(1 row)
```

例

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra](#) (callback function version), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Mean4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

11.13.2 ST_InvDistWeight4ma

[ST_InvDistWeight4ma](#) — 使用反距离加权法 (Inverse Distance Weighted method) 对栅格进行插值的函数。

Synopsis

double precision **ST_InvDistWeight4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

例

[ST_InvDistWeight4ma](#) (Inverse Distance Weighted method) 对栅格进行插值的函数。

userargs 是包含插值参数的文本数组，至少包含 2 个参数。第一个参数是插值权重 (力率) 的指数 (通常记为 k)。第二个参数是插值权重，默认为 1。第三个参数是插值权重，默认为 1。第四个参数是插值权重，默认为 1。第五个参数是插值权重，默认为 1。第六个参数是插值权重，默认为 1。第七个参数是插值权重，默认为 1。第八个参数是插值权重，默认为 1。第九个参数是插值权重，默认为 1。第十个参数是插值权重，默认为 1。第十一个参数是插值权重，默认为 1。第十二个参数是插值权重，默认为 1。第十三个参数是插值权重，默认为 1。第十四个参数是插值权重，默认为 1。第十五个参数是插值权重，默认为 1。第十六个参数是插值权重，默认为 1。第十七个参数是插值权重，默认为 1。第十八个参数是插值权重，默认为 1。第十九个参数是插值权重，默认为 1。第二十个参数是插值权重，默认为 1。第二十个参数是插值权重，默认为 1。

插值权重 k 的取值范围是 $0 \leq k \leq 1$ 。

$$\hat{z}(x_o) = \frac{\sum_{j=1}^m z(x_j) d_{ij}^{-k}}{\sum_{j=1}^m d_{ij}^{-k}}$$

$k =$ 插值权重 (power factor), $0 \leq k \leq 1$



Note

[ST_MapAlgebra](#) (callback function version) 对栅格进行插值的函数。

2.1.0 版本中，[ST_InvDistWeight4ma](#) 函数被弃用。

¶

-- NEEDS EXAMPLE

¶

[ST_MapAlgebra \(callback function version\)](#), [ST_MinDist4ma](#)

11.13.3 ST_Max4ma

ST_Max4ma — 返回 4 个输入栅格中的最大值。

Synopsis

float8 **ST_Max4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Max4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

¶

返回 4 个输入栅格中的最大值。

需要 2 个参数，userargs 可以是 NODATA 或空字符串。



Note

从 1 开始 [ST_MapAlgebraFctNgb](#) 返回最大值。



Note

从 2 开始 [ST_MapAlgebra \(callback function version\)](#) 返回最大值。



Warning

2.1.0 之前 [ST_MapAlgebraFctNgb](#) 返回 1 个最大值。

2.0.0 之前返回最大值。

从 2.1.0 开始返回 2 个最大值。

¶

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_max4ma(float[][],text,text[])':: ↵
    regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
   2 |      254
(1 row)
```



ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

11.13.4 ST_Mean4ma

ST_Mean4ma —

Synopsis

```
float8 ST_Mean4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Mean4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-
args);
```

[illegible]

```

2  ,  userargs  NODATA

```



Note

1 ST MapAlgebraFctNgb



Note

Note 2 **ST_MapAlgebra (callback function version)**



Warning

Warning
2.1.0 ST_MapAlgebraFctNgb 1

2.0.0 ☒☒☒☒☒☒☒☒☒☒.

☒☒☒☒: 2.1.0 ☒☒☒☒☒☒ 2 ☒☒☒☒☒☒☒.

例 1

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_mean4ma(float[][],text,text[])':: ↵
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
rid |      st_value
-----+-----
  2 | 253.222229003906
(1 row)
```

例 2

```
SELECT
  rid,
  st_value(
    ST_MapAlgebra(rast, 1, 'st_mean4ma(double precision[][][], integer[][], text ↵
      [])'::regprocedure,'32BF', 'FIRST', NULL, 1, 1)
    , 2, 2)
FROM dummy_rast
WHERE rid = 2;
rid |      st_value
-----+-----
  2 | 253.222229003906
(1 row)
```

函数

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra](#) (callback function version), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#),
[ST_Range4ma](#), [ST_StdDev4ma](#)

11.13.5 ST_Min4ma

ST_Min4ma — 返回栅格中每个像素的 4 邻域的最小值。

Synopsis

float8 **ST_Min4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Min4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

函数

返回栅格中每个像素的 4 邻域的最小值。

参数 2 是文本，是 userargs 中 NODATA 模式的索引。



Note

从 1 开始 **ST_MapAlgebraFctNgb** 函数将不再支持。.



Note

从 2 开始 **ST_MapAlgebra (callback function version)** 函数将不再支持。.



Warning

2.1.0 版本 **ST_MapAlgebraFctNgb** 函数将不再支持 1 个参数。.

2.0.0 版本 **ST_MapAlgebraFctNgb** 函数将不再支持。
从 2.1.0 版本开始 2 个参数。

SQL

```
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)
```

函数

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 4 个输入栅格中的最小距离 (欧几里得) 值。

Synopsis

double precision **ST_MinDist4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);

注意

ST_InvDistWeight4ma 函数在 2.1.0 版本中引入。



Note

ST_InvDistWeight4ma 函数在 2.1.0 版本中引入。



Note

ST_MapAlgebra (callback function version) 函数在 2.1.0 版本中引入。

2.1.0 版本引入。

注意

-- NEEDS EXAMPLE

注意

ST_MapAlgebra (callback function version), ST_InvDistWeight4ma

11.13.7 ST_Range4ma

ST_Range4ma — 计算 4 邻域的范围。

Synopsis

float8 **ST_Range4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Range4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

注意

ST_Range4ma 函数在 2.1.0 版本中引入。
2 个参数，userargs 参数 NODATA 模式。



Note

1 ST_MapAlgebraFctNgb 函数在 2.1.0 版本中引入。



Note

2 ST_MapAlgebra (callback function version) 函数在 2.1.0 版本中引入。



Warning

2.1.0 版本中 `ST_MapAlgebraFctNgb` 函数已弃用，将在 1 年后被移除。

2.0.0 版本中已弃用。

替代方案：2.1.0 版本中 `ST_MapAlgebra` 函数。

示例

```
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)
```

示例

`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 个移动平均值的标准差。

Synopsis

`float8 ST_StdDev4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);`
`double precision ST_StdDev4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);`

示例

计算 4 个移动平均值的标准差。



Note

从 1 版本开始，`ST_MapAlgebraFctNgb` 函数已弃用。

**Note**

从 2.0.0 开始，`ST_MapAlgebra (callback function version)` 已弃用。请使用 `ST_MapAlgebraFctNgb`。

**Warning**

从 2.1.0 开始，`ST_MapAlgebraFctNgb` 已弃用。请使用 `ST_MapAlgebra`。

2.0.0 版本弃用。

从 2.1.0 版本开始，2 个参数。

SQL

```
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)
```

SQL

`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` — 4 个移动平均值的和。

Synopsis

`float8 ST_Sum4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);`
`double precision ST_Sum4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);`

SQL

从 2.0.0 版本开始，2 个参数。

从 2.1.0 版本开始，2 个参数，`userargs` 参数为 `NODATA` 模式。



Note

从 1 开始 `ST_MapAlgebraFctNgb` 函数使用回调函数。



Note

从 2 开始 `ST_MapAlgebra (callback function version)` 函数使用回调函数。



Warning

2.1.0 版本 `ST_MapAlgebraFctNgb` 函数使用 1 个回调函数。

2.0.0 版本 `ST_MapAlgebraFctNgb` 函数使用 2 个回调函数。
从 2.1.0 版本开始使用 2 个回调函数。

SQL

```
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)
```

SQL

`ST_MapAlgebraFctNgb`, `ST_MapAlgebra (callback function version)`, `ST_Min4ma`, `ST_Max4ma`, `ST_Mean4ma`, `ST_Range4ma`, `ST_Distinct4ma`, `ST_StdDev4ma`

11.14 统计函数

11.14.1 ST_Aspect

`ST_Aspect` — 返回指定栅格的坡度 (度) 值。返回值的范围是 0 到 90 度。

Synopsis

raster **ST_Aspect**(raster rast, integer band=1, text pixeltype=32BF, text units=DEGREES, boolean interpolate_nodata=FALSE);
raster **ST_Aspect**(raster rast, integer band, raster customextent, text pixeltype=32BF, text units=DEGREES, boolean interpolate_nodata=FALSE);

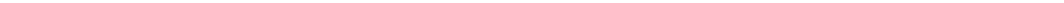

```

WITH foo AS (
    SELECT ST_Tile(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
                1, '32BF', 0, -9999
            ),
            1, 1, 1, ARRAY[
                [1, 1, 1, 1, 1, 1],
                [1, 1, 1, 1, 2, 1],
                [1, 2, 2, 3, 3, 1],
                [1, 1, 3, 2, 1, 1],
                [1, 2, 2, 1, 2, 1],
                [1, 1, 1, 1, 1, 1]
            ]::double precision[]
        ),
        2, 2
    ) AS rast
)
SELECT
    t1.rast,
    ST_Aspect(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;

```


ST MapAlgebra (callback function version), ST TRI, ST TPI, ST Roughness, ST HillShade, ST Slope

11.14.2 ST_HillShade

ST HillShade — 

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);
```

```

interpolate_nodata=FALSE),
raster ST_HillShade(raster rast, integer band, raster customextent, text pixeltype=32BF, double pre-
cision azimuth=315, double precision altitude=45, double precision max_bright=255, double preci-
sion scale=1.0, boolean interpolate nodata=FALSE);

```

[illegible]

azimuth 0 360.

altitude 0 90 (天頂) 0 90

max bright 0 255 255 0 255 255.

`scale` 指定栅格单元的大小。 例如: `ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999), 1, 1, 1, ARRAY[`
`scale=111120` 指定栅格单元的大小。

`interpolate_nodata` 指定是否对 NODATA 值进行插值。 `ST_InvDistWeight4ma` 指定是否对 NODATA 值进行插值。



Note

有关 `ST_Hillshade` 函数的工作原理, 请参见 [How hillshade works](#)。

2.0.0 版本中, `ST_Hillshade` 函数使用 `interpolate_nodata` 参数。

例如: 2.1.0 版本中 `ST_MapAlgebra()` 函数使用 `interpolate_nodata` 参数。

例如: 2.1.0 版本中 `ST_Hillshade` 函数使用 `interpolate_nodata` 参数。 2.1.0 版本中 `ST_Hillshade` 函数使用 `interpolate_nodata` 参数。

例如: 1

```
WITH foo AS (
  SELECT ST_SetValues(
    ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1],
      [1, 2, 2, 2, 1],
      [1, 2, 3, 2, 1],
      [1, 2, 2, 2, 1],
      [1, 1, 1, 1, 1]
    ]::double precision[]
  ) AS rast
)
SELECT
  ST_DumpValues(ST_Hillshade(rast, 1, '32BF'))
FROM foo
```

```
(1,"{NULL,NULL,NULL,NULL,NULL},{NULL,251.32763671875,220.749786376953,147.224319458008,↵
NULL},{NULL,220.749786376953,180.312225341797,67.7497863769531,NULL},{NULL ↵
,147.224319458008
,67.7497863769531,43.1210060119629,NULL},{NULL,NULL,NULL,NULL,NULL}}")
(1 row)
```

例如: 2

例如: PostgreSQL 9.1 版本中, `ST_Hillshade` 函数使用 `interpolate_nodata` 参数。

```
WITH foo AS (
  SELECT ST_Tile(
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
```

```

        1, '32BF', 0, -9999
    ),
    1, 1, 1, ARRAY[
        [1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 2, 1],
        [1, 2, 2, 3, 3, 1],
        [1, 1, 3, 2, 1, 1],
        [1, 2, 2, 1, 2, 1],
        [1, 1, 1, 1, 1, 1]
    ]::double precision[]
),
2, 2
) AS rast
)
SELECT
    t1.rast,
    ST_Hillshade(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;
```



ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_Aspect, ST_Slope

11.14.3 ST_Roughness

ST Roughness — DEM “(roughness)” “.”

Synopsis

```
raster ST_Roughness(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean
interpolate nodata=FALSE );
```



XXXXXXXXXXXXXXXXXXXXXXXXX DEM "XXX" XXXXXXXX.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.



```
-- needs examples
```


ST MapAlgebra (callback function version), ST TRI, ST TPI, ST Slope, ST HillShade, ST Aspect

11.14.4 ST_Slope

ST_Slope — 计算栅格的坡度 (斜率) 值。返回栅格的坡度值。

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);
```

参数

rast 栅格 (raster) 值。返回栅格的坡度值。

units 返回的坡度值的单位。 RADIANS, DEGREES(度), PERCENT 百分比。

scale 返回的坡度值的精度。 例如: 设置 scale=370400, 则: 返回的坡度值 scale=111120 精度。

interpolate_nodata 是否插值, 如果为 TRUE 则使用 **ST_InvDistWeight4ma** 函数插值, 如果为 FALSE 则使用 NODATA 值。



Note

坡度 (slope), 方位 (aspect), 阴影 (hillshade) 函数在 [ESRI - How hillshade works](#) 和 [ERDAS Field Guide - Slope Images](#) 中有详细描述。

2.0.0 版本中已弃用。

更新: 2.1.0 版本中 **ST_MapAlgebra()** 函数已弃用, 使用 **units**, **scale**, **interpolate_nodata** 参数。

更新: 2.1.0 版本中 **ST_Slope()** 函数已弃用。 2.1.0 版本中 **ST_Slope()** 函数已弃用。

示例: 示例 1

```
WITH foo AS (
  SELECT ST_SetValues(
    ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1],
      [1, 2, 2, 2, 1],
      [1, 2, 3, 2, 1],
      [1, 2, 2, 2, 1],
      [1, 1, 1, 1, 1]
    ]::double precision[]
  ) AS rast
)
SELECT
  ST_DumpValues(ST_Slope(rast, 1, '32BF'))
FROM foo

          st_dumpvalues
```

```

(1,"{10.0249881744385,21.5681285858154,26.5650520324707,21.5681285858154,10.0249881744385},{21.5681285858154,26.5650520324707,36.8698959350586,0,36.8698959350586,26.5650520324707},{21.5681285858154,35.26438905681285858154,26.5650520324707,21.5681285858154,10.0249881744385}}")
(1 row)

```

图例: 图 2

图例: 图 2. 图例 PostgreSQL 9.1 图例.

```

WITH foo AS (
  SELECT ST_Tile(
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
        1, '32BF', 0, -9999
      ),
      1, 1, 1, ARRAY[
        [1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 2, 1],
        [1, 2, 2, 3, 3, 1],
        [1, 1, 3, 2, 1, 1],
        [1, 2, 2, 1, 2, 1],
        [1, 1, 1, 1, 1, 1]
      ]::double precision[]
    ),
    2, 2
  ) AS rast
)
SELECT
  t1.rast,
  ST_Slope(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;

```

图例

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_HillShade, ST_Aspect

11.14.5 ST_TPI

ST_TPI — 图例 (Topographic Position Index) 图例.

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
默认 1 (focalmean radius of one) 使用。

2.1.0 版本。

¶

-- needs examples

¶

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) 计算。

Synopsis

raster **ST_TRI**(raster rast, integer nband, raster customextent, text pixeltypes="32BF" , boolean interpolate_nodata=FALSE);

¶

地形崎岖度指数 (Terrain Ruggedness Index) 计算。



Note
默认 1 (focalmean radius of one) 使用。

2.1.0 版本。

¶

-- needs examples

国国

[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);

国国

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

国国

```
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')
)
```

国国

[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);

国国

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

国国

```
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;
```

国国

[ST_InterpolateRaster](#)

11.15 几何函数

11.15.1 Box3D

Box3D — 返回 BOX3D 几何体。

Synopsis

box3d **Box3D**(raster rast);

返回

返回 BOX3D 几何体。

返回 ((MINX, MINY), (MAXX, MAXY)) 定义的 BOX3D。

版本: 2.0.0 引入 BOX3D 函数。BOX2D 函数在 2.0.0 版本中引入 BOX3D 函数。

返回

```
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)

返回

ST_Envelope

11.15.2 ST_ConvexHull

ST_ConvexHull — 返回 BandNoDataValue 的凸包，返回 ST_Envelope 的凸包。

Synopsis

geometry **ST_ConvexHull**(raster rast);

geomval (SRF; Set-Returning Function) geom (geom) val (val) geomval
geomval geomval. geomval val geomval
geomval.

ST_DumpAsPolygon(**geom**) GROUP BY (**geom**)

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

GDAL 1.7 ☒☒☒☒☒☒☒☒☒.



Note!

Note

If there is a no data value set for a band, pixels with that value will not be returned except in the case of `exclude_nodata_value=false`.



Note

, ST ValueCount



Note!

Note

ST PixelAsPolygons



```
-- this syntax requires PostgreSQL 9.3+
SELECT val, ST_AsText(geom) As geomwkt
FROM (
SELECT dp.*
FROM dummy_rast, LATERAL ST_DumpAsPolygons(rast) AS dp
WHERE rid = 2
) As foo
WHERE val BETWEEN 249 and 251
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))



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。如果输入栅格具有 float8 精度，则返回的几何体也具有 float8 精度。

返回的几何体是一个矩形，其边界由 ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)) 定义。

示例

```
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))

示例

ST_Envelope, ST_AsText, ST_SRID

11.15.5 ST_MinConvexHull

ST_MinConvexHull — 返回给定栅格的最小凸包。

Synopsis

geometry **ST_MinConvexHull**(raster rast, integer nband=NULL);

返回

如果输入栅格具有 NODATA 值，则返回的几何体也将具有 NODATA 值。nband 为 NULL 时，返回所有非 NODATA 带的凸包。

2.1.0 版本引入。

图 11.15.5

```
WITH foo AS (
  SELECT
    ST_SetValues(
      ST_SetValues(
        ST_AddBand(ST_MakeEmptyRaster(9, 9, 0, 0, 1, -1, 0, 0, 0), 1, '8 ←
          BUI', 0, 0), 2, '8BUI', 1, 0),
        1, 1, 1,
        ARRAY[
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 1, 0, 0, 0, 0, 1],
          [0, 0, 0, 1, 1, 0, 0, 0, 0],
          [0, 0, 0, 1, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0]
        ]::double precision[][])
      ),
      2, 1, 1,
      ARRAY[
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 0]
      ]::double precision[][])
    ) AS rast
)
SELECT
  ST_AsText(ST_ConvexHull(rast)) AS hull,
  ST_AsText(ST_MinConvexHull(rast)) AS mhull,
  ST_AsText(ST_MinConvexHull(rast, 1)) AS mhull_1,
  ST_AsText(ST_MinConvexHull(rast, 2)) AS mhull_2
FROM foo
```

hull	mhull_1	mhull	mhull_2
POLYGON((0 0,9 0,9 -9,0 -9,0 0))	POLYGON((0 -3,9 -3,9 -9,0 -9,0 -3))	POLYGON((3 -3,9 -3,9 -6,3 -6,3 -3))	POLYGON((0 -3,6 -3,6 -9,0 -9,0 -3))

图 11.15.6

ST_Envelope, ST_ConvexHull, ST_ConvexHull, ST_AsText

11.15.6 ST_Polygon

ST_Polygon — NODATA 返回 NULL。如果输入是 NULL，则返回 NULL。

Synopsis

geometry **ST_Polygon**(raster rast, integer band_num=1);

注意

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

0.1.6 添加。GDAL 1.7 添加。

更新: 2.1.0 添加 (C 添加)。添加。

更新: 2.1.0 添加, 添加。

注意

```
-- by default no data band value is 0 or not set, so polygon will return a square polygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt
-----
MULTIPOLYGON(((3427927.75 5793244,3427928 5793244,3427928 5793243.75,3427927.75 5793243.75,3427927.75 5793244)))

-- now we change the no data value of first band
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,1,254)
WHERE rid = 2;
SELECT rid, ST_BandNoDataValue(rast)
from dummy_rast where rid = 2;

-- ST_Polygon excludes the pixel value 254 and returns a multipolygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt
-----
MULTIPOLYGON(((3427927.9 5793243.95,3427927.85 5793243.95,3427927.85 5793244,3427927.9 5793244,3427927.9 5793243.95)),((3427928 5793243.85,3427928 5793243.8,3427927.95 5793243.8,3427927.95 5793243.85,3427927.9 5793243.85,3427927.9 5793243.9,3427927.9 5793243.95,3427927.95 5793243.95,3427928 5793243.95,3427928 5793243.85)),((3427927.8 5793243.75,3427927.75 5793243.75,3427927.75 5793243.8,3427927.75 5793243.85,3427927.75 5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.9,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85 5793243.8,3427927.85 5793243.75,3427927.8 5793243.75)))

-- Or if you want the no data value different for just one time

SELECT ST_AsText(
    ST_Polygon(
        ST_SetBandNoDataValue(rast,1,252)
    )
) As geomwkt
FROM dummy_rast
WHERE rid =2;
```

```
geomwkt
```

```
-----
MULTIPOLYGON(((3427928 5793243.85,3427928 5793243.8,3427928 5793243.75,3427927.85 ↵
5793243.75,3427927.8 5793243.75,3427927.8 5793243.8,3427927.75 5793243.8,3427927.75 ↵
5793243.85,3427927.75 5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.85 ↵
5793244,3427927.9 5793244,3427928 5793244,3427928 5793243.95,3427928 5793243.85) ↵
,(3427927.9 5793243.9,3427927.9 5793243.85,3427927.95 5793243.85,3427927.95 ↵
5793243.9,3427927.9 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}}
```

函数

ST_MakeEmptyRaster

11.16 栅格函数

11.16.1 &&

&& — A 栅格与 B 栅格/几何图形的交集是否为 TRUE 的布尔值。

Synopsis

```
boolean &&( raster A , raster B );
boolean &&( raster A , geometry B );
boolean &&( geometry B , raster A );
```

函数

&& 返回 A 栅格与 B 栅格/几何图形的交集是否为 TRUE 的布尔值。



Note

如果任一操作数 (operand) 是 NULL，则返回 NULL。

2.0.0 版本引入。

函数

```
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 A 与 B 的交集不为 TRUE，则返回 FALSE。



Note

如果 (operand) 为 NULL，则返回 NULL。

```
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);

> 返回 A 与 B 的并集是否为 TRUE，如果 A 与 B 的并集不为 TRUE，则返回 FALSE。

**Note**

运算符 (operand) 必须都是栅格类型。

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 的对应像素值相等时返回 TRUE。否则返回 FALSE。

Synopsis

```
boolean =( raster A , raster B );
```

例

$=$ 返回栅格 A 与栅格 B 的对应像素值相等时返回 TRUE。PostgreSQL 支持 $=$, $<$, $>$ 运算符 (如: GROUP BY 或 ORDER BY)。

**Caution**

运算符 (operand) 必须都是栅格类型。 $\sim$ 运算符 (group by) 不支持。

2.1.0 版本中支持。

例

$\sim$

11.16.5 @

@ — B 与 A 的交集是否为 TRUE。如果为 TRUE，则返回 TRUE。

Synopsis

```
boolean @( raster A , raster B );
boolean @( geometry A , raster B );
boolean @( raster B , geometry A );
```

返回

@ 返回 B 与 A 的交集是否为 TRUE。



Note

如果 B 是 NULL，则返回 NULL。

2.0.0 添加 raster @ raster, raster @ geometry。

2.0.5 添加 geometry @ raster。

返回

~

11.16.6 ~=

~= — A 与 B 的交集是否为 TRUE。

Synopsis

```
boolean ~= ( raster A , raster B );
```

返回

~= 返回 A 与 B 的交集是否为 TRUE。



Note

如果 B 是 NULL，则返回 NULL。

2.0.0 添加。

例

以下查询将精度为 2 的栅格与精度为 1 的栅格进行比较，并返回精度为 2 的栅格。

```
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 不相交返回 TRUE。否则返回 FALSE。

Synopsis

```
boolean ~( raster A , raster B );
boolean ~( geometry A , raster B );
boolean ~( raster B , geometry A );
```

例

~ 返回 TRUE 当 A 与 B 不相交时。否则返回 FALSE。



Note

~ 的 (operand) 必须是栅格或几何体。

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 的几何体，则返回 TRUE。否则返回 FALSE。

Synopsis

```
boolean ST_Contains( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Contains( raster rastA , raster rastB );
```


`rastA` `rastB`, `rastB` `rastA` `rastA` `rastB`. `NULL`, `(NODATA)`



Note

[illegible]

Note!

Note

ST_Contains(ST_Polygon(raster), geometry) $\Leftrightarrow$ ST_Contains(geometry, ST_Polygon(raster))



Note

ST_Contains() ☒ ST_Within() ☒☒☒☒☒☒☒. ☒☒☒, ST_Contains(rastA, rastB) ☒☒☒☒
ST_Within(rastB, rastA) ☒☒☒☒☒☒☒☒☒☒☒.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒.



```
-- specified band numbers
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↵
    dummy_rast r2 WHERE r1.rid = 1;
```

NOTICE: The first raster provided has no bands

```

rid | rid | st_contains
-----+-----
  1 |  1 |
  1 |  2 | f

```

```
-- no band numbers specified
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ↵
    dummy_rast r2 WHERE r1.rid = 1;
```

```

rid | rid | st_contains
-----+-----
  1 |  1 | t
  1 |  2 | f

```



ST Intersects, ST Within

11.17.2 ST Contains Properly

ST ContainsProperly — rastB $\not\subseteq$ rastA $\wedge \text{rastA} \not\subseteq \text{rastA} \oplus \text{rastB}$

Synopsis

```
boolean ST_ContainsProperly( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_ContainsProperly( raster rastA , raster rastB );
```

注意

如果 rastB 包含 rastA 且 rastA 与 rastB 的交集面积等于 rastA 与 rastB 的面积，则返回 true。如果 rastA 或 rastB 为 NULL，则返回 NULL。如果 rastA 或 rastB 包含 NODATA 值，则返回 NULL。

如果 rastA 与 rastB 的交集面积等于 rastA 的面积，则返回 true。



Note

如果 rastA 与 rastB 的交集面积等于 rastA 的面积，则返回 true。



Note

如果 rastA 与 rastB 的交集面积等于 rastA 的面积，则返回 true。ST_ContainsProperly(ST_Polygon(raster), geometry) 与 ST_ContainsProperly(geometry, ST_Polygon(raster)) 返回 true 当且仅当 ST_Polygon 包含 geometry。

2.1.0 版本引入。

注意

```
SELECT r1.rid, r2.rid, ST_ContainsProperly(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_containsproperly
2	1	f
2	2	f

注意

ST_Intersects, **ST_Contains**

11.17.3 ST_Covers

ST_Covers — 如果 rastB 包含 rastA 且 rastA 与 rastB 的交集面积等于 rastA 的面积，则返回 true。

Synopsis

```
boolean ST_Covers( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Covers( raster rastA , raster rastB );
```

返回

如果 rastB 与 rastA 不相交，则返回 NULL。如果 rastA 与 rastB 不相交，则返回 NULL。如果 rastA 与 rastB 相交，则返回 (NODATA 值)。



Note

如果 rastA 与 rastB 不相交，则返回 NULL。



Note

如果 rastA 与 rastB 不相交，则返回 NULL。如果 rastA 与 rastB 相交，则返回 (NODATA 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 与 rastB 不相交，则返回 NULL。

Synopsis

```
boolean ST_CoveredBy( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_CoveredBy( raster rastA , raster rastB );
```

返回

如果 rastA 与 rastB 不相交，则返回 NULL。如果 rastA 与 rastB 相交，则返回 (NODATA 值)。



Note

ST_CoveredBy(geometry, ST_Polygon(raster)) 与 ST_CoveredBy(ST_Polygon(raster), geometry) 返回的结果是相同的。



Note

ST_CoveredBy(geometry, ST_Polygon(raster)) 与 ST_CoveredBy(ST_Polygon(raster), geometry) 返回的结果是相同的。ST_CoveredBy(geometry, ST_Polygon(raster)) 与 ST_CoveredBy(ST_Polygon(raster), geometry) 返回的结果是相同的。

2.1.0 版本中，ST_CoveredBy(geometry, ST_Polygon(raster)) 与 ST_CoveredBy(ST_Polygon(raster), geometry) 返回的结果是相同的。

SQL

```
SELECT r1.rid, r2.rid, ST_CoveredBy(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_coveredby
2	1	f
2	2	t

SQL

ST_Intersects, ST_Covers

11.17.5 ST_Disjoint

ST_Disjoint — 检查两个栅格是否不相交。

Synopsis

boolean **ST_Disjoint**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Disjoint**(raster rastA , raster rastB);

SQL

ST_Disjoint 检查两个栅格是否不相交。如果两个栅格不相交，则返回 true。如果两个栅格相交，则返回 false。如果两个栅格中有一个是 NULL，则返回 NULL。如果两个栅格都是 NULL，则返回 true。



Note

ST_Disjoint 检查两个栅格是否不相交。

**Note**

ST_Disjoint(ST_Polygon(raster), geometry) 与 ST_Polygon 不同。

2.1.0 版本。

```

-- rid = 1 has no bands, hence the NOTICE and the NULL value for st_disjoint
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
    dummy_rast r2 WHERE r1.rid = 2;

NOTICE: The second raster provided has no bands
rid | rid | st_disjoint
-----+-----+-----
  2 |   1 |
  2 |   2 | f

```

```

-- this time, without specifying band numbers
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN
    dummy_rast r2 WHERE r1.rid = 2;

rid | rid | st_disjoint
-----+-----+-----
  2 |   1 | t
  2 |   2 | f

```

ST_Intersects**11.17.6 ST_Intersects**

ST_Intersects — 检查 rasterA 与 rasterB 是否相交。

Synopsis

boolean **ST_Intersects**(raster rastA , integer nbandA , raster rastB , integer nbandB);

boolean **ST_Intersects**(raster rastA , raster rastB);

boolean **ST_Intersects**(raster rast , integer nband , geometry geommin);

boolean **ST_Intersects**(raster rast , geometry geommin , integer nband=NULL);

boolean **ST_Intersects**(geometry geommin , raster rast , integer nband=NULL);

检查 rasterA 与 rasterB 是否相交。如果任一输入为 NULL 则返回 NULL，如果任一输入为 (NODATA) 则返回 (NODATA)。

**Note**

ST_Intersects(geometry, raster) 和 ST_Intersects(raster, geometry) 是等效的。

PostGIS: 2.0.0 版本引入/支持。

**Warning**

PostGIS: 2.1.0 版本引入 ST_Intersects(geometry, raster) 和 ST_Intersects(raster, geometry) 的别名。

SQL

```
-- different bands of same raster
SELECT ST_Intersects(rast, 2, rast, 3) FROM dummy_rast WHERE rid = 2;

st_intersects
-----
t
```

SQL

[ST_Intersection](#), [ST_Disjoint](#)

11.17.7 ST_Overlaps

ST_Overlaps — 检查 rasterA 是否与 rasterB 重叠。如果重叠，返回 true，否则返回 false。

Synopsis

```
boolean ST_Overlaps( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Overlaps( raster rastA , raster rastB );
```

SQL

检查 rasterA 是否与 rasterB 重叠。如果重叠，返回 true，否则返回 false。如果 rasterA 与 rasterB 重叠，则返回 true。如果任一输入为 NULL，则返回 NULL。如果任一输入为 (NODATA 值)，则返回 false。

**Note**

ST_Overlaps(geometry, raster) 和 ST_Overlaps(raster, geometry) 是等效的。

**Note**

ST_Overlaps(ST_Polygon(raster), geometry) 返回 TRUE 当且仅当 ST_Polygon 与 geometry 重叠。

2.1.0 版本引入。

SQL

```
-- comparing different bands of same raster
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;

st_overlaps
-----
f
```

SQL

ST_Intersects

11.17.8 ST_Touches

ST_Touches — 返回 TRUE 当且仅当 rastA 与 rastB 在边界处接触，但不重叠。

Synopsis

boolean **ST_Touches**(raster rastA , integer nbandA , raster rastB , integer nbandB);
 boolean **ST_Touches**(raster rastA , raster rastB);

SQL

ST_Touches(rastA, nbandA, rastB, nbandB) 返回 TRUE 当且仅当 rastA 与 rastB 在边界处接触，但不重叠。如果任一输入为 NULL，则返回 NULL。如果任一输入为 NO_DATA，则返回 FALSE。

**Note**

ST_Touches 与 ST_Overlaps 类似，但 ST_Touches 只检查边界接触。

**Note**

ST_Touches(ST_Polygon(raster), geometry) 返回 TRUE 当且仅当 ST_Polygon 与 geometry 在边界处接触。

2.1.0 版本引入。

例

```
SELECT r1.rid, r2.rid, ST_Touches(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_touches
2	1	f
2	2	f

例

ST_Intersects

11.17.9 ST_SameAlignment

ST_SameAlignment — 检查两个栅格是否具有相同的对齐方式 (即，它们的像素大小、旋转、偏移量、SRID 是否相同)。如果两个栅格具有相同的对齐方式，则返回 true，否则返回 false。

Synopsis

```
boolean ST_SameAlignment( raster rastA , raster rastB );
boolean ST_SameAlignment( double precision ulx1 , double precision uly1 , double precision scalex1
, double precision scaley1 , double precision skewx1 , double precision skewy1 , double precision ulx2
, double precision uly2 , double precision scalex2 , double precision scaley2 , double precision skewx2
, double precision skewy2 );
boolean ST_SameAlignment( raster set rastfield );
```

例

例 1, 2): (栅格 1, 2): (栅格 1, 2) 具有相同的对齐方式 (即，它们的像素大小、旋转、偏移量、SRID 是否相同)。如果两个栅格具有相同的对齐方式，则返回 true，否则返回 false。

例 3): 检查两个栅格是否具有相同的对齐方式。ST_SameAlignment 是 PostgreSQL 的聚合函数 (aggregate)。它返回 true 或 false。SUM() 或 AVG() 可以用于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;
```

```
sm
----
t
```

```
SELECT ST_SameAlignment(A.rast,b.rast)
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;

NOTICE: The two rasters provided have different SRIDs
NOTICE: The two rasters provided have different SRIDs
st_samealignment
-----
t
f
f
f
```

¶¶

Section [10.1](#), [ST_NotSameAlignmentReason](#), [ST_MakeEmptyRaster](#)

11.17.10 ST_NotSameAlignmentReason

`ST_NotSameAlignmentReason` — 返回两个栅格不匹配的特定原因。返回值为枚举类型。

Synopsis

text `ST_NotSameAlignmentReason`(raster rastA, raster rastB);

¶¶

返回值为枚举类型，可能的值如下：

**Note**

返回值为枚举类型，可能的值如下：

2.1.0 版本引入。

¶¶

```
SELECT
  ST_SameAlignment(
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
  ),
  ST_NotSameAlignmentReason(
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
  )
;

st_samealignment | st_otsamealignmentreason
-----+-----
f                | The rasters have different scales on the X axis
(1 row)
```

¶

Section 10.1, [ST_SameAlignment](#)

11.17.11 ST_Within

ST_Within — 检查 rasterB 是否完全在 rasterA 内部, 检查 rasterA 是否完全在 rasterB 内部。

Synopsis

boolean **ST_Within**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Within**(raster rastA , raster rastB);

¶

检查 rasterB 是否完全在 rasterA 内部, 检查 rasterA 是否完全在 rasterB 内部。如果 rasterA 或 rasterB 包含 NULL 值, 则返回 NULL。如果 rasterA 或 rasterB 包含 (NODATA 值) 则返回 NULL。



Note
检查 (operand) 是否完全在内部。



Note
检查 ST_Within(ST_Polygon(raster), geometry) 或 ST_Within(geometry, ST_Polygon(raster)) 是否返回 ST_Polygon 的结果。



Note
ST_Within() 与 ST_Contains() 相反。检查, ST_Within(rastA, rastB) 与 ST_Contains(rastB, rastA) 相反。

2.1.0 版本。

¶

```
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
```

函数

[ST_Intersects](#), [ST_Contains](#), [ST_DWithin](#), [ST_DFullyWithin](#)

11.17.12 ST_DWithin

`ST_DWithin` — 检查栅格 `rastA` 是否与栅格 `rastB` 在指定距离内相交。

Synopsis

```
boolean ST_DWithin( raster rastA , integer nbandA , raster rastB , integer nbandB , double precision
distance_of_srid );
boolean ST_DWithin( raster rastA , raster rastB , double precision distance_of_srid );
```

函数

检查栅格 `rastA` 是否与栅格 `rastB` 在指定距离内相交。如果任一栅格为 NULL，则返回 NULL。如果任一栅格包含 NODATA 值，则返回 NULL。

距离以 SRID 为单位指定。如果 SRID 未指定，则默认为 4326。



Note

距离 (operand) 必须是非负数。



Note

如果栅格 `rastA` 是 `ST_Polygon`，则 `ST_DWithin(ST_Polygon(raster), geometry)` 与 `ST_DWithin(ST_Polygon(geometry), raster)` 返回相同的结果。

2.1.0 版本引入。

函数

```
SELECT r1.rid, r2.rid, ST_DWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1 CROSS
JOIN dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_dwithin
2	1	f
2	2	t

函数

[ST_Within](#), [ST_DFullyWithin](#)

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。如果 distance_of_srid 是 NULL，则返回 TRUE。如果 distance_of_srid 是 0，则返回 FALSE。如果 distance_of_srid 是正数，则返回 TRUE。如果 distance_of_srid 是负数，则返回 FALSE。

如果 distance_of_srid 是正数，则返回 TRUE。如果 distance_of_srid 是负数，则返回 FALSE。如果 distance_of_srid 是 0，则返回 FALSE。如果 distance_of_srid 是 NULL，则返回 TRUE。



Note

如果 distance_of_srid 是 0，则返回 FALSE。



Note

ST_DFullyWithin(ST_Polygon(raster), geometry) 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.

12.1 ☒☒☒☒☒☒☒

👉👉👉👉 **PAGC standardizer** 👉👉👉 (fork) 👉👉 (👉👉👉👉👉👉👉👉 **PAGC PostgreSQL** 👉👉👉👉👉👉👉👉).

字典(lexicon; lex) 和 词表(gazetteer; gaz) 是自然语言处理中的两个重要概念。

```
CREATE EXTENSION address_standardizer;
PostgreSQL address_standardizer,
address_standardizer_data_us,
address_standardizer_data_us;
CREATE EXTENSION address_standardizer_data_us;
```

PostGIS extensions/address_standardizer                                      

Section 2.3

12.1.1

XXXXXXXXXXXXXXXXXXXXXXXXXXXXX, &/&, XXXXXXXXXXXXXXXX (macro) XXXXXXXXX, X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (micro) XXXXXXXXX
X. XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX, XXXXXXXXXXXXXXX.

□□□□ □□□□□□□□□□/□, □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□.

0000/0000 (**zip code**) 0 (Perl) 00000000000000000000000000000000. 0000000000
 parseaddress-api.c 00000000, 00000000000000000000000000000000.

1/1 (Perl) 编译选项: 编译选项: parseaddress-api.c 编译选项: 编译选项: "includes" 编译选项: 编译选项:

。

。。

id 。

rule 。。 **PAGC Address Standardizer Rule records** 。

， -1, -1, 0 17 。

2 0 2 22 3 -1 5 5 6 7 3 -1 2 6 TYPE NUMBER TYPE
DIRECT QUALIF , STREET STREET SUFTYP SUFDIR QUALIF
6 ARC_C 。

stdaddr。

。

-1 。。 **PAGC Input Tokens** 。

。

AMPERS (13)。 (&) "and" 。

DASH (9)。 (句讀法; punctuation) 。

DOUBLE (21)。 2 。

FRACT (25)。

MIXED (23)。

NUMBER (0)。

ORD (15)。 "First" "1st" 。

ORD (18)。

WORD (1)。

。

BOXH (14)。

BUILDH (19)。

BUILD 。

DIRECT (22)。

MILE (20)。

ROAD (6)。

RR (8)。

TYPE (2)。

UNITH (16). 4文字の単位。例: *APT* の *UNIT*

4文字の単位

QUINT (28). 5文字の単位。例: (Zip Code) 5文字の単位。

QUAD (29). 4文字の単位。ZIP4 4文字の単位。

PCH (27). 例, 例, 4文字の単位。3 文字の単位。例: FSA 4文字の単位。

PCT (26). 例, 例, 4文字の単位。3 文字の単位。例: LDU 4文字の単位。

STOPWORD (不用語; **stopword**)

STOPWORD 例 **WORD** 4文字の単位。例: **WORD** 例 **STOPWORD** 4文字の単位 **WORD** 4文字の単位。

STOPWORD (7). 4文字の単位。例: *THE*

4文字の単位

例 -1(例) 4文字の単位, 4文字の単位 -1 4文字の単位。 **stdaddr** 4文字の単位。 the section called “**4文字の単位**” 4文字の単位。

4文字の単位

4文字の単位。 4文字の単位 (例) 0 例 (例) 17 4文字の単位。

MACRO_C

(例 = “0”). *PLACE STATE ZIP* 例 **MACRO** 4文字の単位。

MACRO_C output tokens (excerpted from <http://www.pgcgeo.org/docs/html/pagc-12.html#--r-typ-->).

CITY (例“10”). 例: “Albany”

STATE (例“11”). 例: “NY”

NATION (例“12”). 例: “USA”

POSTAL (例“13”). (SADS 例“ZIP CODE”, “PLUS 4”). 4文字の単位。

MICRO_C

(例 = “1”). (例, 例, sufdir, predir, pretyp, suftype, qualif 例) 例 **MICRO** 4文字の単位 (例: **ARC_C** 例 **CIVIC_C**). 4文字の単位。

MICRO_C output tokens (excerpted from <http://www.pgcgeo.org/docs/html/pagc-12.html#--r-typ-->).

HOUSE 例 (例 1) 例: 75 State Street 例 75 例

predir 例 (例 2) 例: North, South, East, West 4文字の単位 (STREET NAME PRE-DIRECTIONAL) 4文字の単位。

qual 例 (例 3) 例: 3715 OLD HIGHWAY 99 例 *OLD*

12.1.3.3 gaz table

gaz table — 地理数据库 (gaz) 表, 包含 (1) 原始输入 (the section called “**raw**”) 和 (2) 标准化后的表示。例如，如果你在美国，你可以加载这些州名及其缩写。

表

A gaz (short for gazetteer) table is used to standardize place names and associate that input with the section called “**raw**” and (b) standardized representations. For example if you are in US, you may load these with State Names and associated abbreviations.

表。包含 (1) 原始输入。包含 (2) 标准化后的表示。

id 整数

seq 整数: 索引? - 索引

word 文本: 单词

stdword 文本: 标准单词

token 文本: 令牌。包含 (1) 原始输入。包含 (2) 标准化后的表示。 **PAGC Tokens** 表。

12.1.4 地理数据库

12.1.4.1 debug_standardize_address

debug_standardize_address — Returns a json formatted text listing the parse tokens and standardizations

Synopsis

text **debug_standardize_address**(text lextab, text gaztab, text rultab, text micro, text macro=NULL);

表

This is a function for debugging address standardizer rules and lex/gaz mappings. It returns a json formatted text that includes the matching rules, mapping of tokens, and best standardized address **stdaddr** form of an input address utilizing **lex table** table name, **gaz table**, and **rules table** table names and an address.

For single line addresses use just micro

For two line address A micro consisting of standard first line of postal address e.g. house_num street, and a macro consisting of standard postal second line of an address e.g city, state postal_code country.

Elements returned in the json document are

input_tokens For each word in the input address, returns the position of the word, token categorization of the word, and the standard word it is mapped to. Note that for some input words, you might get back multiple records because some inputs can be categorized as more than one thing.

rules The set of rules matching the input and the corresponding score for each. The first rule (highest scoring) is what is used for standardization

stdaddr The standardized address elements **stdaddr** that would be returned when running **standardize_address**

Availability: 3.4.0

 This method needs address_standardizer extension.

☒☒

address_standardizer_data_us ☒☒☒☒☒☒☒☒

```
CREATE EXTENSION address_standardizer_data_us; -- only needs to be done once
```

Variant 1: Single line address and returning the input tokens

```
SELECT it->>'pos' AS position, it->>'word' AS word, it->>'stdword' AS standardized_word,
       it->>'token' AS token, it->>'token-code' AS token_code
FROM jsonb(
    debug_standardize_address('us_lex',
                              'us_gaz', 'us_rules', 'One Devonshire Place, PH 301, Boston, MA 02109')
    ) AS s, jsonb_array_elements(s->'input_tokens') AS it;
```

position	word	standardized_word	token	token_code
0	ONE	1	NUMBER	0
0	ONE	1	WORD	1
1	DEVONSHIRE	DEVONSHIRE	WORD	1
2	PLACE	PLACE	TYPE	2
3	PH	PATH	TYPE	2
3	PH	PENTHOUSE	UNITT	17
4	301	301	NUMBER	0
(7 rows)				

Variant 2: Multi line address and returning first rule input mappings and score

```
SELECT (s->'rules'->0->>'score')::numeric AS score, it->>'pos' AS position,
       it->>'input-word' AS word, it->>'input-token' AS input_token, it->>'mapped-word' AS ↵
       standardized_word,
       it->>'output-token' AS output_token
FROM jsonb(
    debug_standardize_address('us_lex',
                              'us_gaz', 'us_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109')
    ) AS s, jsonb_array_elements(s->'rules'->0->'rule_tokens') AS it;
```

score	position	word	input_token	standardized_word	output_token
0.876250	0	ONE	NUMBER	1	HOUSE
0.876250	1	DEVONSHIRE	WORD	DEVONSHIRE	STREET
0.876250	2	PLACE	TYPE	PLACE	SUFTYP
0.876250	3	PH	UNITT	PENTHOUSE	UNITT
0.876250	4	301	NUMBER	301	UNITT
(5 rows)					

☒☒

stdaddr, **rules table**, **lex table**, **gaz table**, **Pagc_Normalize_Address**

12.1.4.2 parse_address

```
parse_address — .
```

Synopsis

```
record parse_address(text address);
```



Returns takes an address as input, and returns a record output consisting of fields *num*, *street*, *street2*, *address1*, *city*, *state*, *zip*, *zipplus*, *country*.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒.

✔ This method needs address_standardizer extension.



☐ ☐ ☐ ☐

```
SELECT num, street, city, zip, zipplus
       FROM parse address('1 Devonshire Place, Boston, MA 02109-1234') AS a;
```

```
num |      street      | city | zip | zipplus
-----+-----+-----+-----+-----
  1 | Devonshire Place | Boston | 02109 | 1234
```

☐ ☐ ☐ ☐ ☐

```
-- basic table
CREATE TABLE places(addid serial PRIMARY KEY, address text);

INSERT INTO places(address)
VALUES ('529 Main Street, Boston MA, 02129'),
       ('77 Massachusetts Avenue, Cambridge, MA 02139'),
       ('25 Wizard of Oz, Walaford, KS 99912323'),
       ('26 Capen Street, Medford, MA'),
       ('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
       ('950 Main Street, Worcester, MA 01610');

-- parse the addresses
-- if you want all fields you can use (a).*
SELECT addid, (a).num, (a).street, (a).city, (a).state, (a).zip, (a).zipplus
FROM (SELECT addid, parse_address(address) As a
      FROM places) AS p;
```

addid	num	street	city	state	zip	zipplus
1	529	Main Street	Boston	MA	02129	323
2	77	Massachusetts Avenue	Cambridge	MA	02139	
3	25	Wizard of Oz	Walaford	KS	99912	
4	26	Capen Street	Medford	MA		
5	124	Mount Auburn St	Cambridge	MA	02138	
6	950	Main Street	Worcester	MA	01610	
(6 rows)						

key	value
box	
city	BOSTON
name	DEVONSHIRE
qual	
unit	# PENTHOUSE 301
extra	
state	MA
predir	
sufdir	
country	USA
pretype	
suftype	PL
building	
postcode	02109
house_num	1
ruralroute	

(16 rows)

图 2: 地址规范化结果。

```
SELECT (each(hstore(p))).*
FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
    'tiger.pagc_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109, US') As p;
```

key	value
box	
city	BOSTON
name	DEVONSHIRE
qual	
unit	# PENTHOUSE 301
extra	
state	MA
predir	
sufdir	
country	USA
pretype	
suftype	PL
building	
postcode	02109
house_num	1
ruralroute	

(16 rows)

图

`stdaddr`, `rules table`, `lex table`, `gaz table`, `Pagc_Normalize_Address`

12.2 TIGER 数据

TIGER 数据是 PostGIS 3.6.1 版本中引入的，用于地址规范化。它包含了一个名为 `tiger` 的表，其中包含了许多用于地址规范化的数据。

- **Nominatim** 是 OpenStreetMap 数据的一个接口。它使用 osm2pgsql 工具，将 OpenStreetMap 数据导入 PostgreSQL 8.4 或 PostgreSQL 9.0 数据库。Nominatim 使用 PostgreSQL 9.0 数据库，并支持 PostgreSQL 9.0 的全文索引。TIGER 数据是一个美国地址数据库，它包含美国的所有地址。Nominatim 使用 TIGER 数据来提供地址搜索功能。Nominatim 使用 SQL 查询来检索数据，并返回 JSON 格式的结果。
- **GIS Graphy** 是一个 PostGIS 的接口，它使用 OSM(OpenStreetMap) 数据。OSM 数据是一个开放街图数据，它包含全球的所有街道。Nominatim 使用 OSM 数据来提供地址搜索功能。Nominatim 使用 Java 1.5, Servlet apps, Solr 等技术。GIS Graphy 是一个 GIS 接口，它使用 PostGIS 数据库来提供 GIS 功能。

12.2.1 Drop_Indexes_Generate_Script

Drop Indexes Generate Script — TIGER 数据是一个美国地址数据库，它包含美国的所有地址。Nominatim 使用 TIGER 数据来提供地址搜索功能。Nominatim 使用 SQL 查询来检索数据，并返回 JSON 格式的结果。

Synopsis

text **Drop_Indexes_Generate_Script**(text param_schema=tiger_data);

返回

TIGER 数据是一个美国地址数据库，它包含美国的所有地址。Nominatim 使用 TIGER 数据来提供地址搜索功能。Nominatim 使用 SQL 查询来检索数据，并返回 JSON 格式的结果。

安装 (bloat) 数据。Install Missing Indexes 数据是一个美国地址数据库，它包含美国的所有地址。Nominatim 使用 TIGER 数据来提供地址搜索功能。Nominatim 使用 SQL 查询来检索数据，并返回 JSON 格式的结果。

2.0.0 版本。

返回

```
SELECT drop_indexes_generate_script() As actionsql;
actionsql
-----
DROP INDEX tiger.idx_tiger_countysub_lookup_lower_name;
DROP INDEX tiger.idx_tiger_edges_countyfp;
DROP INDEX tiger.idx_tiger_faces_countyfp;
DROP INDEX tiger.tiger_place_the_geom_gist;
DROP INDEX tiger.tiger_edges_the_geom_gist;
DROP INDEX tiger.tiger_state_the_geom_gist;
DROP INDEX tiger.idx_tiger_addr_least_address;
DROP INDEX tiger.idx_tiger_addr_tlid;
DROP INDEX tiger.idx_tiger_addr_zip;
DROP INDEX tiger.idx_tiger_county_countyfp;
DROP INDEX tiger.idx_tiger_county_lookup_lower_name;
DROP INDEX tiger.idx_tiger_county_lookup_snd_name;
DROP INDEX tiger.idx_tiger_county_lower_name;
DROP INDEX tiger.idx_tiger_county_snd_name;
DROP INDEX tiger.idx_tiger_county_the_geom_gist;
DROP INDEX tiger.idx_tiger_countysub_lookup_snd_name;
DROP INDEX tiger.idx_tiger_cousub_countyfp;
DROP INDEX tiger.idx_tiger_cousub_cousubfp;
DROP INDEX tiger.idx_tiger_cousub_lower_name;
DROP INDEX tiger.idx_tiger_cousub_snd_name;
```

```

DROP INDEX tiger.idx_tiger_cousub_the_geom_gist;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_least_address;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_tlid;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_zip;
DROP INDEX tiger_data.idx_tiger_data_ma_county_countyfp;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_snd_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_snd_name;
:
:

```

安装

[Install_Missing_Indexes, Missing_Indexes_Generate_Script](#)

12.2.2 Drop_Nation_Tables_Generate_Script

Drop_Nation_Tables_Generate_Script — 删除 county_all, state_all 表，并生成 county, state 表 (州) 的索引。

Synopsis

text **Drop_Nation_Tables_Generate_Script**(text param_schema=tiger_data);

安装

删除 county_all, state_all 表，并生成 county, state 表 (州) 的索引。tiger_2010 和 tiger_2011 表的索引。

2.1.0 版本。

安装

```

SELECT drop_nation_tables_generate_script();
DROP TABLE tiger_data.county_all;
DROP TABLE tiger_data.county_all_lookup;
DROP TABLE tiger_data.state_all;
DROP TABLE tiger_data.ma_county;
DROP TABLE tiger_data.ma_state;

```

安装

[Loader_Generate_Nation_Script](#)

12.2.3 Drop_State_Tables_Generate_Script

Drop_State_Tables_Generate_Script — 删除 (州) 的索引，并生成 tiger_data 表的索引。

Synopsis

text **Drop_State_Tables_Generate_Script**(text param_state, text param_schema=tiger_data);

说明

该函数用于生成删除指定州 (州) 的 tiger_data 数据库表的 SQL 脚本。该脚本将删除指定州 (州) 的所有 tiger_data 数据库表。该脚本将删除指定州 (州) 的所有 tiger_data 数据库表。

2.0.0 版本新增。

示例

```
SELECT drop_state_tables_generate_script('PA');
DROP TABLE tiger_data.pa_addr;
DROP TABLE tiger_data.pa_county;
DROP TABLE tiger_data.pa_county_lookup;
DROP TABLE tiger_data.pa_cousub;
DROP TABLE tiger_data.pa_edges;
DROP TABLE tiger_data.pa_faces;
DROP TABLE tiger_data.pa_featnames;
DROP TABLE tiger_data.pa_place;
DROP TABLE tiger_data.pa_state;
DROP TABLE tiger_data.pa_zip_lookup_base;
DROP TABLE tiger_data.pa_zip_state;
DROP TABLE tiger_data.pa_zip_state_loc;
```

说明

Loader_Generate_Script

12.2.4 Geocode

Geocode — 将给定的地址 (地址) 转换为 NAD83 地理坐标, 并返回该地址的地理坐标。该函数将给定的地址 (地址) 转换为 NAD83 地理坐标, 并返回该地址的地理坐标。该函数将给定的地址 (地址) 转换为 NAD83 地理坐标, 并返回该地址的地理坐标。

Synopsis

setof record **geocode**(varchar address, integer max_results=10, geometry restrict_region=NULL, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);
setof record **geocode**(norm_addy in_addy, integer max_results=10, geometry restrict_region=NULL, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);

安装

PostGIS (PostgreSQL 扩展) 依赖于 NAD83 坐标系, 因此需要安装 `normalized_address (addy)` 扩展。该扩展依赖于 `tiger` 扩展。Tiger 扩展 (edge, face, addr), PostgreSQL 扩展 (soundex, levenshtein), 以及 PostGIS 扩展 (tiger) 都需要安装。安装 Tiger 扩展 (edge/face) 需要 10 分钟。

安装: 2.0.0 版本的 Tiger 2010 扩展, 需要安装 PostgreSQL 9.1rc1/PostGIS 2.0 扩展。3.0 GHz 处理器 2GB 内存 7 分钟。安装 `max_results` 扩展。

安装

安装 (MA), 安装 (MN), 安装 (CA), 安装 (RI) 需要 Tiger 扩展 PostgreSQL 9.1rc1/PostGIS 2.0 扩展。3.0 GHz 处理器 2GB 内存 7 分钟。安装 `max_results` 扩展。

安装 (61 分钟)。

```
SELECT g.rating, ST_X(g.geomout) As lon, ST_Y(g.geomout) As lat,
       (addy).address As stno, (addy).streetname As street,
       (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,( ←
       addy).zip
FROM geocode('75 State Street, Boston MA 02109', 1) As g;
rating |          lon          |          lat          | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | -71.0557505845646 | 42.35897920691 | 75 | State | St | Boston | MA | 02109
```

安装 (122 ~ 150 分钟)。

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
       (addy).address As stno, (addy).streetname As street,
       (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,( ←
       addy).zip
FROM geocode('226 Hanover Street, Boston, MA',1) As g;
rating |          wktlonlat          | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----
1 | POINT(-71.05528 42.36316) | 226 | Hanover | St | Boston | MA | 02113
```

安装 (500 分钟)。

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
       (addy).address As stno, (addy).streetname As street,
       (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,( ←
       addy).zip
FROM geocode('31 - 37 Stewart Street, Boston, MA 02116',1) As g;
rating |          wktlonlat          | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----
70 | POINT(-71.06466 42.35114) | 31 | Stuart | St | Boston | MA | 02116
```

安装 (batch) 扩展。max_results = 1 扩展。安装 (batch) 扩展。

```
CREATE TABLE addresses_to_geocode(addid serial PRIMARY KEY, address text,
                                   lon numeric, lat numeric, new_address text, rating integer);
```

```
INSERT INTO addresses_to_geocode(address)
VALUES ('529 Main Street, Boston MA, 02129'),
```

```

('77 Massachusetts Avenue, Cambridge, MA 02139'),
('25 Wizard of Oz, Walaford, KS 99912323'),
('26 Capen Street, Medford, MA'),
('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
('950 Main Street, Worcester, MA 01610');

-- only update the first 3 addresses (323-704 ms - there are caching and shared memory ↵
-- effects so first geocode you do is always slower) --
-- for large numbers of addresses you don't want to update all at once
-- since the whole geocode must commit at once
-- For this example we rejoin with LEFT JOIN
-- and set to rating to -1 rating if no match
-- to ensure we don't regeocode a bad address
UPDATE addresses_to_geocode
  SET (rating, new_address, lon, lat)
    = ( COALESCE(g.rating, -1), pprint_addy(g.addy),
        ST_X(g.geomout)::numeric(8,5), ST_Y(g.geomout)::numeric(8,5) )
FROM (SELECT addid, address
      FROM addresses_to_geocode
      WHERE rating IS NULL ORDER BY addid LIMIT 3) As a
  LEFT JOIN LATERAL geocode(a.address,1) As g ON true
WHERE a.addid = addresses_to_geocode.addid;

result
-----
Query returned successfully: 3 rows affected, 480 ms execution time.

SELECT * FROM addresses_to_geocode WHERE rating is not null;

addid |          address          | lon | lat | ↵
-----+-----+-----+-----+-----
1 | 529 Main Street, Boston MA, 02129 | -71.07177 | 42.38357 | 529 Main St, ↵
  | Boston, MA 02129 | 0
2 | 77 Massachusetts Avenue, Cambridge, MA 02139 | -71.09396 | 42.35961 | 77 ↵
  | Massachusetts Ave, Cambridge, MA 02139 | 0
3 | 25 Wizard of Oz, Walaford, KS 99912323 | -97.92913 | 38.12717 | Willowbrook, ↵
  | KS 67502 | 108
(3 rows)

```

练习: 练习

```

SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
  (addy).address As stno, (addy).streetname As street,
  (addy).streettypeabbrev As styp,
  (addy).location As city, (addy).stateabbrev As st,(addy).zip
FROM geocode('100 Federal Street, MA',
  3,
  (SELECT ST_Union(the_geom)
   FROM place WHERE statefp = '25' AND name = 'Lynn')::geometry
 ) As g;

rating |          wktlonlat          | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----
7 | POINT(-70.96796 42.4659) | 100 | Federal | St | Lynn | MA | 01905
16 | POINT(-70.96786 42.46853) | NULL | Federal | St | Lynn | MA | 01905
(2 rows)

```

Time: 622.939 ms



Normalize Address, Pprint Addy, ST AsText, ST SnapToGrid, ST X, ST Y

12.2.5 Geocode_Intersection

Geocode_Intersection — geom_out , $\text{normalized\_address}(\text{addy})$. TIGER (edge, face, addr) PostgreSQL (soundex, levenshtein).

Synopsis

```
setof record geocode_intersection(text roadway1, text roadway2, text in_state, text in_city, text
in_zip, integer max_results=10, norm addy OUT addy, geometry OUT geomout, integer OUT rating);
```


Normalized address (addy), NAD83 geomout, (rating) PostgreSQL (soundex, levenshtein)

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.



数据库 (MA) 使用 TIGER 数据 PostgreSQL 9.0/PostGIS 1.5 数据库 3.0 GHZ 内存 2GB 硬盘 7 块 (3,000 GB).

TIGER 2011 PostGIS 2.0 PostgreSQL 64 8GB 2003 64 (41).

```
SELECT pprint_addy(addy), st_astext(geomout),rating
      FROM geocode_intersection( 'Haverford St','Germania St', 'MA', 'Boston', '02130',1);
```

pprint_addy	st_astext	rating
98 Haverford St, Boston, MA 02130	POINT(-71.101375 42.31376)	0

64 7 3,500 , 2003 741 .

```
SELECT pprint_addy(addy), st_astext(geomout),rating
      FROM geocode_intersection('Weld', 'School', 'MA', 'Boston');
      pprint addy          |          st_astext          | rating
```

Address	Point	Ordering
98 Weld Ave, Boston, MA 02119	POINT(-71.099 42.314234)	3
99 Weld Ave, Boston, MA 02119	POINT(-71.099 42.314234)	3

函数

[Set_Geocode_Setting](#)

12.2.7 Get_Tract

Get Tract — 返回指定地理坐标 (tract) 的地理ID (field) 的文本值。该函数返回的地理ID是NAD83坐标系下的。

Synopsis

text **get_tract**(geometry loc_geom, text output_field=name);

函数

该函数返回指定地理坐标 (tract) 的地理ID (field) 的文本值。该函数返回的地理ID是NAD83坐标系下的。

Note

This function uses the census tract which is not loaded by default. If you have already loaded your state table, you can load tract as well as bg, and tabblock using the [Loader_Generate_Census_Script](#) script.



If you have not loaded your state data yet and want these additional tables loaded, do the following

```
UPDATE tiger.loader_lookuptables SET load = true WHERE load = false AND lookup_name IN('tract', 'bg', 'tabblock');
```

then they will be included by the [Loader_Generate_Script](#).

2.0.0 版本引入。

函数

```
SELECT get_tract(ST_Point(-71.101375, 42.31376) ) As tract_name;
tract_name
-----
1203.01
```

```
--this one returns the tiger geoid
SELECT get_tract(ST_Point(-71.101375, 42.31376), 'tract_id' ) As tract_id;
tract_id
-----
25025120301
```

函数

[Geocode](#)>

12.2.8 Install_Missing_Indexes

Install Missing Indexes —    (join)  (key)                           

Synopsis

```
boolean Install_Missing_Indexes();
```

[illegible]

```
SELECT install_missing_indexes();
install_missing_indexes
-----
t
```

11

Loader Generate Script, Missing Indexes Generate Script

12.2.9 Loader Generate Census Script

Loader_Generate_Census_Script — 州, TIGER (州) (tract), (bg), (tabblock) tiger_data . (州) .

Synopsis

```
setof text loader_generate_census_script(text[] param states, text os);
```



州) 州), TIGER 州) tract, bg, tabblocks 州)
 tiger_data 州). 州) 州)
 州).

解压缩 (使用 7-zip) 文件, 使用 wget 下载。
 参考 Section 4.7.2 中的描述。
 将文件复制到 (州) 目录。
 将文件命名为 "staging" 或 "temp"。

OS _____.

1. `loader_variables` - 设置环境变量, 例如, `loader_variables` (staging) 数据库连接字符串。
2. `loader_platform` - 设置加载平台, 例如, `loader_platform`。设置加载平台, 例如, `loader_platform`。
3. `loader_lookuptables` - 设置加载表 (表, 表), 例如, `loader_lookuptables`, 例如, `loader_lookuptables`。设置加载表, 例如, `loader_lookuptables`, 例如, `loader_lookuptables`。设置加载表 (州名) 表, TIGER 加载表, 例如, `loader_lookuptables`。例如: `tiger.faces` 加载表 `tiger_data.ma_faces` 加载表。

2.0.0 安装与配置。



Note

Loader_Generate_Script 安装与配置, PostGIS 2.0.0 alpha5 安装与配置 TIGER 加载表, 例如, `loader_generate_script` (州) 加载表。

安装

安装与配置。

```
SELECT loader_generate_census_script(ARRAY['MA'], 'windows');
-- result --
set STATEDIR="\gisdata\www2.census.gov\geo\pvs\tiger2010st\25_Massachusetts"
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\projects\pg\pg91win\bin\
set PGPORT=5432
set PGHOST=localhost
set PGUSER=postgres
set PGPASSWORD=yourpasswordhere
set PGDATABASE=tiger_postgis20
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata

%WGETTOOL% http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent -- \
    relative --accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
del %TMPDIR%\*. * /Q
%PSQL% -c "DROP SCHEMA tiger_staging CASCADE;"
%PSQL% -c "CREATE SCHEMA tiger_staging;"
cd %STATEDIR%
for /r %%z in (*.zip) do %UNZIPTOOL% e %%z -o%TMPDIR%
cd %TMPDIR%
%PSQL% -c "CREATE TABLE tiger_data.MA_tract(CONSTRAINT pk_MA_tract PRIMARY KEY (tract_id) ) \
    INHERITS(tiger.tract); "
%SHP2PGSQL% -c -s 4269 -g the_geom -W "latin1" tl_2010_25_tract10.dbf tiger_staging. \
    ma_tract10 | %PSQL%
%PSQL% -c "ALTER TABLE tiger_staging.MA_tract10 RENAME geoid10 TO tract_id; SELECT \
    loader_load_staged_data(lower('MA_tract10'), lower('MA_tract')); "
%PSQL% -c "CREATE INDEX tiger_data_MA_tract_the_geom_gist ON tiger_data.MA_tract USING gist \
    (the_geom);"
%PSQL% -c "VACUUM ANALYZE tiger_data.MA_tract;"
%PSQL% -c "ALTER TABLE tiger_data.MA_tract ADD CONSTRAINT chk_statefp CHECK (statefp = \
    '25');"
:
```


.sh 安装脚本。

```
STATEDIR="/gisdata/www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts"
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
WGETTOOL="/usr/bin/wget"
export PGBIN=/usr/pgsql-9.0/bin
export PGPORT=5432
export PGHOST=localhost
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata

wget http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent --relative ←
--accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
rm -f ${TMPDIR}/*. *
${PSQL} -c "DROP SCHEMA tiger_staging CASCADE;"
${PSQL} -c "CREATE SCHEMA tiger_staging;"
cd $STATEDIR
for z in *.zip; do $UNZIPTOOL -o -d $TMPDIR $z; done
:
:
```

。

Loader_Generate_Script

12.2.10 Loader_Generate_Script

Loader_Generate_Script — 生成加载脚本 (州) 的 TIGER 数据 tiger_data 的加载脚本。 (州) 的 TIGER 2010 数据, 加载脚本, 加载脚本。

Synopsis

setof text **loader_generate_script**(text[] param_states, text os);

。

生成加载脚本 (州) 的 TIGER 数据 tiger_data 的加载脚本。 (州) 的 TIGER 2010 数据。

使用 unzip (7-zip) 和 wget 脚本。 参考 Section 4.7.2。 生成加载脚本 (州) 的 TIGER 2010 数据。 生成“staging” “temp” 脚本。

生成 OS 脚本。

1. loader_variables - 加载脚本, 加载脚本 (staging) 脚本。

2. loader_platform - 加载平台信息。返回一个字典，包含平台名称、版本、架构等信息。
3. loader_lookuptables - 加载查找表 (字典)，用于快速查找数据。返回一个字典，包含查找表的名称、版本、架构等信息。

2.0.0  TIGER 2010  (tract),  (bg),  (tabblock) .



Note

If you are using pgAdmin 3, be warned that by default pgAdmin 3 truncates long text. To fix, change *File -> Options -> Query Tool -> Query Editor -> Max. characters per column* to larger than 50000 characters.



Using psql where gistest is your database and /gisdata/data_load.sh is the file to create with the shell commands to run.

```
psql -U postgres -h localhost -d gistest -A -t \  
-c "SELECT Loader Generate Script(ARRAY['MA'], 'gistest')" > /gisdata/data/load.sh;
```

□ (州) 2 □□□□□□□□□□□□□□□□□□□□□□□□□□□□.

```
SELECT loader_generate_script(ARRAY['MA','RI'], 'windows') AS result;
-- result --
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\Program Files\PostgreSQL\9.4\bin\
set PGPORT=5432
set PGHOST=localhost
set PGUSER=postgres
set PGPASSWORD=yourpasswordhere
set PGDATABASE=geocoder
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata

cd \gisdata
%WGETTOOL% ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl_*_25_* --no-parent --relative ↵
--recursive --level=2 --accept=zip --mirror --reject=html
cd \gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE
:
:
```

.sh [redacted].

```
SELECT loader_generate_script(ARRAY['MA','RI'], 'sh') AS result;
-- result --
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
WGETTOOL="/usr/bin/wget"
export PGBIN=/usr/lib/postgresql/9.4/bin
-- variables used by psql: https://www.postgresql.org/docs/current/static/libpq-envvars.html
export PGPOR=5432
```

```

export PGHOST=localhost
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata

cd /gisdata
wget ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl_*_25_* --no-parent --relative --recursive --level=2 --accept=zip --mirror --reject=html
cd /gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE
rm -f ${TMPDIR}/*. *
:
:

```

❏

Section 2.4.1, [Pprint_Addy](#), [ST_AsText](#)

12.2.11 Loader_Generate_Nation_Script

Loader_Generate_Nation_Script — 生成加载国家数据的脚本，用于加载国家数据。

Synopsis

text **loader_generate_nation_script**(text os);

❏

该脚本生成一个名为 `loader_generate_nation_script` 的脚本，用于加载国家数据。该脚本将生成一个名为 `loader_generate_nation_script` 的脚本，用于加载国家数据。该脚本将生成一个名为 `loader_generate_nation_script` 的脚本，用于加载国家数据。

该脚本使用 `unzip` 命令（需要 7-zip 支持）来解压国家数据。该脚本使用 `wget` 命令来下载国家数据。该脚本使用 `unzip` 命令（需要 7-zip 支持）来解压国家数据。

该脚本在 OS 上使用 `loader_generate_nation_script` 来生成国家数据。该脚本在 OS 上使用 `loader_generate_nation_script` 来生成国家数据。该脚本在 OS 上使用 `loader_generate_nation_script` 来生成国家数据。

1. `loader_variables` - 生成国家数据的脚本，用于加载国家数据。该脚本将生成一个名为 `loader_variables` 的脚本，用于加载国家数据。
2. `loader_platform` - 生成国家数据的脚本，用于加载国家数据。该脚本将生成一个名为 `loader_platform` 的脚本，用于加载国家数据。
3. `loader_lookuptables` - 生成国家数据的脚本，用于加载国家数据。该脚本将生成一个名为 `loader_lookuptables` 的脚本，用于加载国家数据。

Enhanced: 2.4.1 zip code 5 tabulation area (zcta5) load step was fixed and when enabled, zcta5 data is loaded as a single table called `zcta5_all` as part of the nation script load.

2.1.0 生成国家数据的脚本。

Note



Note!

If you want zip code 5 tabulation area (zcta5) to be included in your nation script load, do the following:

```
UPDATE tiger.loader_lookuptables SET load = true WHERE table_name = 'zcta510';
```

Note

Note!

tiger_2010 表 (州) tiger_2011 表, [Drop_Nation_Tables_Generate_Script](#) 脚本.

[illegible]

```
SELECT loader_generate_nation_script('windows');
```

```
SELECT loader_generate_nation_script('sh');
```



Loader Generate Script, Missing Indexes Generate Script

12.2.12 Missing_Indexes_Generate_Script

Missing Indexes Generate Script — `@@@join` (join) `@@@key` (key) `@@@`
`@@@` SQL DDL `@@@`.

Synopsis

```
text Missing_Indexes_Generate_Script();
```

[illegible]

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

❏

```
SELECT missing_indexes_generate_script();
-- output: This was run on a database that was created before many corrections were made to ←
the loading script ---
CREATE INDEX idx_tiger_county_countyfp ON tiger.county USING btree(countyfp);
CREATE INDEX idx_tiger_cousub_countyfp ON tiger.cousub USING btree(countyfp);
CREATE INDEX idx_tiger_edges_tfidl ON tiger.edges USING btree(tfidl);
CREATE INDEX idx_tiger_edges_tfidl ON tiger.edges USING btree(tfidl);
CREATE INDEX idx_tiger_zip_lookup_all_zip ON tiger.zip_lookup_all USING btree(zip);
CREATE INDEX idx_tiger_data_ma_county_countyfp ON tiger_data.ma_county USING btree(countyfp ←
);
CREATE INDEX idx_tiger_data_ma_cousub_countyfp ON tiger_data.ma_cousub USING btree(countyfp ←
);
CREATE INDEX idx_tiger_data_ma_edges_countyfp ON tiger_data.ma_edges USING btree(countyfp);
CREATE INDEX idx_tiger_data_ma_faces_countyfp ON tiger_data.ma_faces USING btree(countyfp);
```

❏

[Loader_Generate_Script, Install_Missing_Indexes](#)

12.2.13 Normalize_Address

Normalize Address — 将地址规范化为标准的格式，并返回一个包含规范化后的地址的表。该表包含以下列：norm_addy、tiger_geocoder、tiger (TIGER 数据表中的表名) 和 address。

Synopsis

norm_addy **normalize_address**(varchar in_address);

❏

该函数将输入的地址规范化为标准的格式，并返回一个包含规范化后的地址的表。该表包含以下列：norm_addy、tiger_geocoder、tiger (TIGER 数据表中的表名) 和 address。

该函数使用 tiger 数据表中的 tiger_geocoder 表来规范化地址。该表包含以下列：norm_addy、tiger_geocoder、tiger (TIGER 数据表中的表名) 和 address。

该函数使用 tiger 数据表中的 tiger 表来规范化地址。

该函数返回一个包含规范化后的地址的表。该表包含以下列：norm_addy、tiger_geocoder、tiger (TIGER 数据表中的表名) 和 address。

(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [state-Abbrev] [zip] [parsed] [zip4] [address_alphanumeric]

Enhanced: 2.4.0 norm_addy object includes additional fields zip4 and address_alphanumeric.

1. address 类型: 字符串。
2. predirAbbrev 类型 varchar: N, S, E, W 表示方向。direction_look 类型 varchar: 表示方向。

3. streetName varchar(255).
4. streetTypeAbbrev varchar(255), St, Ave, Cir 等。street_type_lookup 表。
5. postdirAbbrev varchar(255), N, S, E, W 等。direction_lookup 表。
6. internal varchar(255)。
7. location varchar(255)。
8. stateAbbrev varchar(255), MA, NY, MI 等 (州名)。state_lookup 表。
9. zip varchar(255)。02109 等。
10. parsed boolean。normalize_address 函数。
11. zip4 last 4 digits of a 9 digit zip code. Availability: PostGIS 2.4.0.
12. address_alphanumeric Full street number even if it has alpha characters like 17R. Parsing of this is better using [PgNormalizeAddress](#) function. Availability: PostGIS 2.4.0.

例

使用 `Pprint_Addy` 函数。

```
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
FROM (SELECT address, normalize_address(address) As na
      FROM addresses_to_geocode) As g;
```

orig	streetname	streettypeabbrev
28 Capen Street, Medford, MA	Capen	St
124 Mount Auburn St, Cambridge, Massachusetts 02138	Mount Auburn	St
950 Main Street, Worcester, MA 01610	Main	St
529 Main Street, Boston MA, 02129	Main	St
77 Massachusetts Avenue, Cambridge, MA 02139	Massachusetts	Ave
25 Wizard of Oz, Walford, KS 99912323	Wizard of Oz	

例

[Geocode](#), [Pprint_Addy](#)

12.2.14 PgNormalizeAddress

`PgNormalizeAddress` — 规范化地址，将地址转换为标准格式，并返回规范化后的地址。使用 `norm_addy` 函数。使用 `tiger_geocoder` 表 (TIGER 地理数据) 进行地址标准化。

Synopsis

```
norm_addy pgc_normalize_address(varchar in_address);
```

##

使用 norm_addy 函数，将地址标准化。该函数返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。

使用 tiger 函数，将地址标准化。该函数返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。

使用 tiger_geocoder 函数，将地址标准化。该函数返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。

使用 norm_addy 函数，将地址标准化。该函数返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。

Normalize_Address 函数返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。

2.1.0 版本更新。

 This method needs address_standardizer extension.

(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip]

使用 address_standardizer 函数，将地址标准化。该函数返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。

house_num, predir, name, suftype, sufdir, unit, city, state, postcode

Enhanced: 2.4.0 norm_addy object includes additional fields zip4 and address_alphanumeric.

1. address 返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。
2. predirAbbrev 返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。
3. streetName 返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。
4. streetTypeAbbrev 返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。
5. postdirAbbrev 返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。
6. internal 返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。
7. location 返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。
8. stateAbbrev 返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。
9. zip 返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。
10. parsed (boolean) 返回一个包含标准化地址的数组。该数组包含以下字段：house_num, predir, name, suftype, sufdir, unit, city, state, postcode。
11. zip4 last 4 digits of a 9 digit zip code. Availability: PostGIS 2.4.0.
12. address_alphanumeric Full street number even if it has alpha characters like 17R. Parsing of this is better using **Pagc_Normalize_Address** function. Availability: PostGIS 2.4.0.

address	predirabbrev	streetname	streettypeabbrev	postdirabbrev	internal	
location	stateabbrev	zip	parsed			
9000	E	R00	ST		SUITE 999	
SPRINGFIELD	CO		t			

```
WITH g AS (SELECT address, ROW((sa).house_num, (sa).predir, (sa).name
, (sa).suftype, (sa).sufdir, (sa).unit , (sa).city, (sa).state, (sa).postcode, true)::
norm_addy As na
FROM (SELECT address, standardize_address('tiger.pagc_lex'
, 'tiger.pagc_gaz'
, 'tiger.pagc_rules', address) As sa
FROM addresses_to_geocode) As g)
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
FROM g;
```

orig	streetname	streettypeabbrev
529 Main Street, Boston MA, 02129	MAIN	ST
77 Massachusetts Avenue, Cambridge, MA 02139	MASSACHUSETTS	AVE
25 Wizard of Oz, Walford, KS 99912323	WIZARD OF	
26 Capen Street, Medford, MA	CAPEN	ST
124 Mount Auburn St, Cambridge, Massachusetts 02138	MOUNT AUBURN	ST
950 Main Street, Worcester, MA 01610	MAIN	ST

```
varchar pprint_addy(norm addy in addy);
```


¶

`norm_addy` 函数返回一个 `geometry` 类型的地址，该地址是输入地址的规范化版本。该函数返回的地址是输入地址的规范化版本。

该函数 `Normalize_Address` 函数返回一个 `geometry` 类型的地址。

¶

该函数返回一个 `geometry` 类型的地址。

```
SELECT pprint_addy(normalize_address('202 East Fremont Street, Las Vegas, Nevada 89101'))
  As pretty_address;
      pretty_address
-----
202 E Fremont St, Las Vegas, NV 89101
```

该函数返回一个 `geometry` 类型的地址。

```
SELECT address As orig, pprint_addy(normalize_address(address)) As pretty_address
  FROM addresses_to_geocode;
```

orig	pretty_address
529 Main Street, Boston MA, 02129	529 Main St, Boston MA, 02129
77 Massachusetts Avenue, Cambridge, MA 02139	77 Massachusetts Ave, Cambridge, MA 02139
28 Capen Street, Medford, MA	28 Capen St, Medford, MA
124 Mount Auburn St, Cambridge, Massachusetts 02138	124 Mount Auburn St, Cambridge, MA 02138
950 Main Street, Worcester, MA 01610	950 Main St, Worcester, MA 01610

¶

`Normalize_Address`

12.2.16 Reverse_Geocode

`Reverse_Geocode` — 该函数返回一个 `geometry` 类型的地址，该地址是输入地址的规范化版本。该函数返回的地址是输入地址的规范化版本。

Synopsis

record **Reverse_Geocode**(geometry pt, boolean include_strnum_range=false, geometry[] OUT intpt, norm_addy[] OUT addy, varchar[] OUT street);

¶

该函数返回一个 `geometry` 类型的地址，该地址是输入地址的规范化版本。该函数返回的地址是输入地址的规范化版本。

`strnum` include strnum range [0..9].
[0..9] are the only valid characters.

st1	st2	st3	cross_str
5 Bradford St, Boston, MA 02118	49 Waltham St, Boston, MA 02118		Waltham St

Geocode 2

```
SELECT actual_addr, lon, lat, pprint_addy((rg).addy[1]) As int_addr1,
       (rg).street[1] As cross1, (rg).street[2] As cross2
FROM (SELECT address As actual_addr, lon, lat,
       reverse_geocode( ST_SetSRID(ST_Point(lon,lat),4326) ) As rg
      FROM addresses_to_geocode WHERE rating
> -1) As foo;
```

actual_addr	lon	lat	↔
int_addr1		cross1	↔
cross2			
529 Main Street, Boston MA, 02129	-71.07181	42.38359	527 Main St, ↔
Boston, MA 02129 Medford St			
77 Massachusetts Avenue, Cambridge, MA 02139	-71.09428	42.35988	77 ↔
Massachusetts Ave, Cambridge, MA 02139 Vassar St			
26 Capen Street, Medford, MA	-71.12377	42.41101	9 Edison Ave, ↔
Medford, MA 02155 Capen St			
124 Mount Auburn St, Cambridge, Massachusetts 02138	-71.12304	42.37328	3 University ↔
Rd, Cambridge, MA 02138 Mount Auburn St			
950 Main Street, Worcester, MA 01610	-71.82368	42.24956	3 Maywood St, ↔
Worcester, MA 01603 Main St			
	Maywood Pl		



Pprint Addy, Pprint Addy, ST AsText

12.2.17 Topology_Load_Tiger

Topology_Load_Tiger — PostGIS TIGER TIGER

Synopsis

```
text Topology Load Tiger(varchar topo name, varchar region type, varchar region id);
```



PostGIS 2.1.1 TIGER 2010 Census Data. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 83

[illegible]

**Note**

安装 TIGER 数据到 PostGIS 数据库。请参考 [Chapter 9 Section 2.2.3](#) 进行安装。安装完成后，需要运行 `shp2pgsql` 命令来加载数据。安装完成后，需要运行 `shp2pgsql` 命令来加载数据。

**Note**

安装 TIGER 数据到 PostGIS 数据库。请参考 [Chapter 9 Section 2.2.3](#) 进行安装。安装完成后，需要运行 `shp2pgsql` 命令来加载数据。安装完成后，需要运行 `shp2pgsql` 命令来加载数据。

安装:

1. `topo_name` - 安装 TIGER 数据到 PostGIS 数据库。
2. `region_type` - 安装 TIGER 数据到 PostGIS 数据库。安装 `place` 和 `county` 数据。安装完成后，需要运行 `shp2pgsql` 命令来加载数据。安装完成后，需要运行 `shp2pgsql` 命令来加载数据。
3. `region_id` - TIGER 数据 ID(geoid) 安装 TIGER 数据到 PostGIS 数据库。安装完成后，需要运行 `shp2pgsql` 命令来加载数据。安装完成后，需要运行 `shp2pgsql` 命令来加载数据。

2.0.0 安装 TIGER 数据。

安装: 安装 TIGER 数据。

安装 TIGER 数据 (2249) 安装 TIGER 数据到 PostGIS 数据库。安装完成后，需要运行 `shp2pgsql` 命令来加载数据。安装完成后，需要运行 `shp2pgsql` 命令来加载数据。

```
SELECT topology.CreateTopology('topo_boston', 2249, 0.25);
createtopology
-----
15
-- 60,902 ms ~ 1 minute on windows 7 desktop running 9.1 (with 5 states tiger data loaded)
SELECT tiger.topology_load_tiger('topo_boston', 'place', '2507000');
-- topology_loader_tiger --
29722 edges holding in temporary. 11108 faces added. 1875 edges of faces added. 20576 nodes added.
19962 nodes contained in a face. 0 edge start end corrected. 31597 edges added.

-- 41 ms --
SELECT topology.TopologySummary('topo_boston');
-- topologysummary--
Topology topo_boston (15), SRID 2249, precision 0.25
20576 nodes, 31597 edges, 11109 faces, 0 topogeoms in 0 layers

-- 28,797 ms to validate yeh returned no errors --
SELECT * FROM
  topology.ValidateTopology('topo_boston');

error      | id1      | id2
-----+-----+-----
```

注意: 在加载 TIGER 数据之前, 必须先加载 boston 数据。

在加载 TIGER 数据之前, 必须先加载 boston 数据。 (26986) 的精度为 0.25 度, 加载 TIGER 数据, 加载, 加载。

```
SELECT topology.CreateTopology('topo_suffolk', 26986, 0.25);
-- this took 56,275 ms ~ 1 minute on Windows 7 32-bit with 5 states of tiger loaded
-- must have been warmed up after loading boston
SELECT tiger.topology_load_tiger('topo_suffolk', 'county', '25025');
-- topology_loader_tiger --
36003 edges holding in temporary. 13518 faces added. 2172 edges of faces added.
24761 nodes added. 24075 nodes contained in a face. 0 edge start end corrected. 38175 ↔
edges added.
-- 31 ms --
SELECT topology.TopologySummary('topo_suffolk');
-- topologysummary--
Topology topo_suffolk (14), SRID 26986, precision 0.25
24761 nodes, 38175 edges, 13519 faces, 0 topogeoms in 0 layers

-- 33,606 ms to validate --
SELECT * FROM
    topology.ValidateTopology('topo_suffolk');
```

error	id1	id2
coincident nodes	81045651	81064553
edge crosses node	81045651	85737793
edge crosses node	81045651	85742215
edge crosses node	81045651	620628939
edge crosses node	81064553	85697815
edge crosses node	81064553	85728168
edge crosses node	81064553	85733413

注意

CreateTopology, CreateTopoGeom, TopologySummary, ValidateTopology

12.2.18 Set_Geocode_Setting

Set_Geocode_Setting — 设置地理编码设置。

Synopsis

text Set_Geocode_Setting(text setting_name, text setting_value);

注意

tiger.geocode_settings 表存储地理编码设置。使用 Get_Geocode_Setting 函数可以获取设置值。

2.1.0 版本新增。

注意: 安装时

安装时 **Geocode** 选项, NOTICE 消息会显示。

```
SELECT set_geocode_setting('debug_geocode_address', 'true') As result;
result
-----
true
```

注意

Get_Geocode_Setting

Chapter 13

PostGIS Special Functions Index

13.1 PostGIS Aggregate Functions

The functions below are spatial aggregate functions that are used in the same way as SQL aggregate function such as sum and average.

- **CG_3DUnion** - Perform 3D union using postgis_sfcgal.
 - **ST_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
 - **ST_3DUnion** - Perform 3D union.
 - **ST_AsFlatGeobuf** - Return a FlatGeobuf representation of a set of rows.
 - **ST_AsGeobuf** - Return a Geobuf representation of a set of rows.
 - **ST_AsMVT** - Aggregate function returning a MVT representation of a set of rows.
 - **ST_ClusterDBSCAN** - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
 - **ST_ClusterIntersecting** - Aggregate function that clusters input geometries into connected sets.
 - **ST_ClusterIntersectingWin** - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
 - **ST_ClusterKMeans** - Window function that returns a cluster id for each input geometry using the K-means algorithm.
 - **ST_ClusterWithin** - Aggregate function that clusters geometries by separation distance.
 - **ST_ClusterWithinWin** - Window function that returns a cluster id for each input geometry, clustering using separation distance.
 - **ST_Collect** - Creates a GeometryCollection or Multi* geometry from a set of geometries.
 - **ST_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。
- **ST_MemUnion** - Aggregate function which unions geometries in a memory-efficient but slower way.
- **ST_Polygonize** - Computes a collection of polygons formed from the linework of a set of geometries.
- **ST_SameAlignment** - 检查两个多边形是否具有相同的对齐方式 (即它们的边是否平行)。如果两个多边形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。
- **TopoElementArray_Agg** - Returns a topoelementarray for a set of element_id, type arrays (topoelements).

13.2 PostGIS Window Functions

The functions below are spatial window functions that are used in the same way as SQL window functions such as `row_number()`, `lead()`, and `lag()`. They must be followed by an `OVER()` clause.

- **ST_ClusterDBSCAN** - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- **ST_ClusterIntersectingWin** - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
- **ST_ClusterKMeans** - Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST_ClusterWithinWin** - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- **ST_CoverageClean** - Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.
- **ST_CoverageInvalidEdges** - Window function that finds locations where polygons fail to form a valid coverage.
- **ST_CoverageSimplify** - Window function that simplifies the edges of a polygonal coverage.

13.3 PostGIS SQL-MM Compliant Functions

The functions given below are PostGIS functions that conform to the SQL/MM 3 standard

- **CG_3DArea** - 3D 面积函数。返回 3D 几何体的面积。如果输入为 null，则返回 null。
- **CG_3DDifference** - 3D 差函数。返回两个 3D 几何体的差集。
- **CG_3DIntersection** - 3D 交集函数。返回两个 3D 几何体的交集。
- **CG_3DUnion** - 3D 并集函数。返回两个 3D 几何体的并集。
- **CG_Volume** - 3D 体积函数。返回 3D 几何体的体积。如果输入为 null，则返回 null。
- **ST_3DArea** - 3D 面积函数。返回 3D 几何体的面积。如果输入为 null，则返回 null。

- **ST_3DDWithin** - Tests if two 3D geometries are within a given 3D distance
- **ST_3DDifference** - 3 `Geometry`.
- **ST_3DDistance** - `Geometry`, `Geometry` (SRS `Geometry`) 3 `Geometry`.
- **ST_3DIntersection** - 3 `Geometry`.
- **ST_3DIntersects** - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- **ST_3DLength** - `Geometry`.
- **ST_3DPerimeter** - `Geometry`.
- **ST_3DUnion** - Perform 3D union.
- **ST_AddEdgeModFace** - `Geometry`, `Geometry`, `Geometry`.
- **ST_AddEdgeNewFaces** - `Geometry`, `Geometry`, `Geometry` 2 `Geometry`.
- **ST_AddIsoEdge** - `Geometry` anode `Geometry` anothernode `Geometry` alinestring `Geometry` ID `Geometry`.
- **ST_AddIsoNode** - `Geometry` (isolated) `Geometry` ID `Geometry`. `Geometry` NULL `Geometry`, `Geometry`.
- **ST_Area** - `Geometry`.
- **ST_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST_AsGML** - `Geometry` GML 2 `Geometry` GML 3 `Geometry`.
- **ST_AsText** - `Geometry`/ `Geometry` WKT(Well-Known Text) `Geometry` SRID `Geometry`.
- **ST_Boundary** - `Geometry`.
- **ST_Buffer** - Computes a geometry covering all points within a given distance from a geometry.
- **ST_Centroid** - `Geometry`.
- **ST_ChangeEdgeGeom** - `Geometry`.
- **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 `Geometry`.
- **ST_CreateTopoGeo** - `Geometry` `Geometry` `Geometry`.
- **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 `Geometry`.

- **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** - ST_LineString 或 ST_CircularString 的终点。
- **ST_Envelope** - 返回两个几何体的包围盒 (double precision; float8)。
- **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 几何体到 ST_Geometry 的转换。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 几何体到 ST_Geometry 的转换。
- **ST_GeomFromWKB** - WKB(Well-Known Binary) 几何体到 ST_Geometry 的转换。SRID 默认为 0。
- **ST_GeometryFromText** - WKT(Well-Known Text) 几何体到 ST_Geometry 的转换。使用 ST_GeomFromText 函数。
- **ST_GeometryN** - ST_Geometry 的 N 个几何体。
- **ST_GeometryType** - ST_Geometry 的几何体类型。
- **ST_GetFaceEdges** - 返回面几何体的边。
- **ST_GetFaceGeometry** - 返回面几何体的 ID 几何体。
- **ST_InitTopoGeo** - Creates a new topology schema and registers it in the topology.topology table.
- **ST_InteriorRingN** - 返回几何体的 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。使用 ST_IsClosed 函数。
- **ST_IsEmpty** - Tests if a geometry is empty.
- **ST_IsRing** - Tests if a LineString is closed and simple.
- **ST_IsSimple** - 几何体是否是简单的。TRUE 表示是。使用 ST_IsSimple 函数。
- **ST_IsValid** - Tests if a geometry is well-formed in 2D.
- **ST_Length** - 返回几何体的长度。
- **ST_LineFromText** - 使用 SRID 和 WKT 创建 LINESTRING。SRID 默认为 0。
- **ST_LineFromWKB** - 使用 SRID 和 WKB 创建 LINESTRING。
- **ST_LinestringFromWKB** - 使用 SRID 和 WKB 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 `ST_MultiLineString` 函数。
- **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** - 修改边，将边分割成两部分，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** - 修改边，将边分割成两部分，并返回新的边 ID。
- **ST_NumCurves** - Return the number of component curves in a CompoundCurve.
- **ST_NumGeometries** - 返回几何体数量。
- **ST_NumInteriorRings** - 返回内部环数量。
- **ST_NumPatches** - 返回补丁数量。如果为 NULL 则返回 0。
- **ST_NumPoints** - ST_LineString 或 ST_CircularString 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 的补丁 N。
- **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** - 从 SRID 和 WKT 文本创建点。SRID 为 0 则默认为 0。
- **ST_PointFromWKB** - 从 SRID 和 WKB 创建点。
- **ST_PointN** - ST_LineString 或 ST_CircularString 的第 N 个点。
- **ST_PointOnSurface** - Computes a point guaranteed to lie in a polygon, or on a geometry.
- **ST_Polygon** - Creates a Polygon from a LineString with a specified SRID.
- **ST_PolygonFromText** - Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.
- **ST_Relate** - Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix

- **ST_RemEdgeModFace** - Removes an edge, and if the edge separates two faces deletes one face and modifies the other face to cover the space of both.
- **ST_RemEdgeNewFace** - 移除边并创建新面。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_RemoveIsoEdge** - Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown.
- **ST_RemoveIsoNode** - 移除孤立节点并返回描述。如果节点不是孤立的，则抛出异常。
- **ST_SRID** - Returns the spatial reference identifier for a geometry.
- **ST_StartPoint** - Returns the first point of a LineString.
- **ST_SymDifference** - Computes a geometry representing the portions of geometries A and B that do not intersect.
- **ST_Touches** - Tests if two geometries have at least one point in common, but their interiors do not intersect.
- **ST_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST_Union** - Computes a geometry representing the point-set union of the input geometries.
- **ST_Volume** - 3D 几何体的体积。对于非 3D 几何体返回 0。
- **ST_WKBToSQL** - WKB(Well-Known Binary) 格式到 ST_Geometry 格式的转换。需要指定 SRID。
- **ST_WKTToSQL** - WKT(Well-Known Text) 格式到 ST_Geometry 格式的转换。
- **ST_Within** - Tests if every point of A lies in B, and their interiors have a point in common.
- **ST_X** - Returns the X coordinate of a Point.
- **ST_Y** - Returns the Y coordinate of a Point.
- **ST_Z** - Returns the Z coordinate of a Point.
- **ST_SRID** - Returns the spatial reference identifier for a topogeometry.

13.4 PostGIS Geography Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a **geography** data type object.

Note



Functions with a (T) are not native geodetic functions, and use a ST_Transform call to and from geometry to do the operation. As a result, they may not behave as expected when going over dateline, poles, and for large geometries or geometry pairs that cover more than one UTM zone. Basic transform - (favoring UTM, Lambert Azimuthal (North/South), and falling back on mercator in worst case scenario)

- **ST_Area** - 计算地理实体的面积。

- **ST_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST_AsEWKT** - 返回 WKT(Well-Known Text) 格式的 SRID 几何/地理数据。
- **ST_AsGML** - 返回 GML 2 或 GML 3 格式的几何/地理数据。
- **ST_AsGeoJSON** - Return a geometry or feature in GeoJSON format.
- **ST_AsKML** - 返回 GML 2 或 GML 3 格式的几何/地理数据。
- **ST_AsSVG** - Returns SVG path data for a geometry.
- **ST_AsText** - 返回 WKT(Well-Known Text) 格式的 SRID 几何/地理数据。
- **ST_Azimuth** - 返回从第一个几何到第二个几何的方位角。
- **ST_Buffer** - Computes a geometry covering all points within a given distance from a geometry.
- **ST_Centroid** - 返回几何/地理数据的质心。
- **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/地理数据之间的最短距离 (longest) 距离。
- **ST_GeogFromText** - WKT (几何) 转换为地理数据。
- **ST_GeogFromWKB** - WKB 转换为地理数据 (EWKB(几何 WKB) 转换为地理数据)。
- **ST_GeographyFromText** - WKT (几何) 转换为地理数据。
- **=** - 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/地理数据的长度。
- **ST_LineInterpolatePoint** - Returns a point interpolated along a line at a fractional location.
- **ST_LineInterpolatePoints** - Returns points interpolated along a line at a fractional interval.
- **ST_LineLocatePoint** - Returns the fractional location of the closest point on a line to a point.
- **ST_LineSubstring** - Returns the part of a line between two fractional locations.
- **ST_Perimeter** - Returns the length of the boundary of a polygonal geometry or geography.
- **ST_Project** - Returns a point projected from a start point by a distance and bearing (azimuth).
- **ST_Segmentize** - Returns a modified geometry/geography having no segment longer than a given distance.
- **ST_ShortestLine** - 返回两个几何/地理数据之间的最短距离。
- **ST_Summary** - 返回几何/地理数据的摘要。
- **<->** - A 与 B 之间的最短距离。
- **&&** - A 与 B 在 2D 空间中是否重合 (TRUE 表示重合)。

13.5 PostGIS Raster Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a **raster** data type object. Listed in alphabetical order.

- **Box3D** - 返回 BOX3D 几何体。
- **@** - B 与 A 相等返回 TRUE。否则返回 FALSE。
- **~** - A 与 B 不相等返回 TRUE。否则返回 FALSE。
- **=** - A 与 B 相等返回 TRUE。否则返回 FALSE。
- **&&** - A 与 B 相等返回 TRUE。否则返回 FALSE。
- **&<** - A 与 B 相等返回 TRUE。否则返回 FALSE。
- **&>** - A 与 B 相等返回 TRUE。否则返回 FALSE。
- **~=** - A 与 B 不相等返回 TRUE。否则返回 FALSE。
- **ST_Retile** - 将栅格重新分块，返回新的栅格。
- **ST_AddBand** - 向栅格添加新的波段 (n) 返回新的栅格。n 可以是波段编号、波段名称、波段名称和波段类型。
- **ST_AsBinary/ST_AsWKB** - 返回栅格的 Well-Known Binary (WKB) 表示。
- **ST_AsGDALRaster** - 返回栅格在指定的 GDAL 栅格格式中的表示。栅格格式是那些由你的编译库支持的。使用 ST_GDALDrivers() 来获取支持的格式列表。
- **ST_AsHexWKB** - 返回栅格的 Well-Known Binary (WKB) 在 Hex 表示。
- **ST_AsJPEG** - 将栅格转换为 JPEG (Joint Photographic Experts Group) 格式 (栅格格式) 返回字节数组。栅格格式可以是 1, 3 或 4。1 表示 RGB 格式，3 表示 RGB 格式，4 表示 RGBA 格式。
- **ST_AsPNG** - 将栅格转换为 PNG (Portable Network Graphics) 格式 (栅格格式) 返回字节数组。栅格格式可以是 1, 3, 4 或 4。1 表示 RGB 格式，3 表示 RGB 格式，4 表示 RGBA 格式。
- **ST_AsRaster** - 将 PostGIS 栅格转换为 PostGIS 栅格。
- **ST_AsRasterAgg** - 聚合。将 PostGIS 几何体聚合为新的栅格。
- **ST_AsTIFF** - 返回栅格选定的波段作为单个 TIFF 图像 (字节数组)。如果没有指定波段或指定的波段不存在于栅格中，则尝试使用所有波段。
- **ST_Aspect** - 返回栅格的坡度 (度) 返回栅格。坡度可以是度、分、秒。
- **ST_Band** - 返回栅格的指定波段。如果没有指定波段，则返回第一个波段。
- **ST_BandFileSize** - 返回栅格波段在文件系统中的文件大小。如果没有指定波段，则假设波段 1。
- **ST_BandFileTimestamp** - 返回栅格波段在文件系统中的文件时间戳。如果没有指定波段，则假设波段 1。
- **ST_BandIsNoData** - 返回栅格波段是否包含 NODATA 值。
- **ST_BandMetaData** - 返回栅格波段的元数据。如果没有指定波段，则返回第一个波段的元数据。

- **ST_BandNoDataValue** - 返回栅格的 NODATA 值。如果栅格没有 NODATA 值，则返回 1。
- **ST_BandPath** - 返回栅格的波段路径。bandnum 指定要返回的波段编号，从 1 开始。
- **ST_BandPixelType** - 返回栅格的波段像素类型。bandnum 指定要返回的波段编号，从 1 开始。
- **ST_Clip** - Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.
- **ST_ColorMap** - 返回栅格的 8BUI 颜色映射 (grayscale, RGB, RGBA) 的 4 个波段。如果指定了 bandnum，则返回指定的波段。
- **ST_Contains** - 返回栅格 rastA 是否包含栅格 rastB。如果 rastB 包含 rastA，则返回 TRUE。
- **ST_ContainsProperly** - rastB 是否完全包含 rastA。如果 rastB 完全包含 rastA，则返回 TRUE。
- **ST_Contour** - Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.
- **ST_ConvexHull** - BandNoDataValue 返回栅格的 NODATA 值，ST_Envelope 返回栅格的包络线。如果指定了 bandnum，则返回指定的波段。
- **ST_Count** - 返回栅格中指定值的像素数量。exclude_nodata_value 指定是否排除 NODATA 值。
- **ST_CountAgg** - 返回栅格中指定值的像素数量。exclude_nodata_value 指定是否排除 NODATA 值。
- **ST_CoveredBy** - 返回栅格 rastA 是否被栅格 rastB 覆盖。
- **ST_Covers** - 返回栅格 rastB 是否覆盖栅格 rastA。
- **ST_DFullyWithin** - 返回栅格 rastA 是否完全在栅格 rastB 的指定距离内。
- **ST_DWithin** - 返回栅格 rastA 是否在栅格 rastB 的指定距离内。
- **ST_Disjoint** - 返回栅格 rastA 是否与栅格 rastB 不相交。
- **ST_DumpAsPolygons** - 返回栅格的几何值 (geom, val) 的集合。如果指定了 bandnum，则返回指定的波段。
- **ST_DumpValues** - 返回栅格的 2 个波段值。
- **ST_Envelope** - 返回栅格的包络线。
- **ST_FromGDALRaster** - 从 GDAL 栅格创建 PostGIS 栅格。
- **ST_GeoReference** - 返回 (world) 的地理参考信息。GDAL 返回 ESRI 的地理参考信息。
- **ST_Grayscale** - Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue
- **ST_HasNoBand** - 返回栅格是否具有指定的波段。如果指定了 bandnum，则返回 TRUE。

- **ST_Height** - 返回栅格的最大高度值。
- **ST_HillShade** - 根据栅格数据生成地形阴影图。参数包括：栅格、阴影比例、阴影方向、阴影颜色。
- **ST_Histogram** - 返回栅格的直方图。参数包括：栅格、bin数、bin范围。
- **ST_InterpolateRaster** - Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation.
- **ST_Intersection** - 返回两个栅格的交集。参数包括：栅格A、栅格B。
- **ST_IntersectionFractions** - Calculates the fraction of each raster cell that is covered by a given geometry.
- **ST_Intersects** - 返回栅格A和栅格B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** - 返回空栅格。参数包括：宽度、高度、SRID、scalex、scaley、skewx、skewy。
- **ST_MapAlgebra (callback function version)** - 返回栅格A和栅格B的映射代数结果。参数包括：栅格A、栅格B、回调函数。
- **ST_MapAlgebraExpr** - 返回栅格A和栅格B的映射代数结果。参数包括：栅格A、栅格B、表达式。
- **ST_MapAlgebraExpr** - 返回栅格A和栅格B的映射代数结果。参数包括：栅格A、栅格B、表达式、extenttype。
- **ST_MapAlgebraFct** - 返回栅格A和栅格B的映射代数结果。参数包括：栅格A、栅格B、函数名。
- **ST_MapAlgebraFct** - 返回栅格A和栅格B的映射代数结果。参数包括：栅格A、栅格B、函数名、extenttype。
- **ST_MapAlgebraFctNgb** - 返回栅格A和栅格B的映射代数结果。参数包括：栅格A、栅格B、函数名、neighborhood。
- **ST_MapAlgebra (expression version)** - 返回栅格A和栅格B的映射代数结果。参数包括：栅格A、栅格B、表达式。
- **ST_MemSize** - 返回栅格的内存大小。
- **ST_MetaData** - 返回栅格的元数据。参数包括：栅格、skew。
- **ST_MinConvexHull** - 返回栅格的最小凸包。

- **ST_NearestValue** - columnx rowy, 返回指定栅格中最近的非空值。如果指定位置为 NODATA 则返回 NODATA。
- **ST_Neighborhood** - columnx rowy, 返回指定栅格中指定位置的邻域。如果指定位置为 NODATA 则返回 NODATA。邻域大小由 2 个参数指定。
- **ST_NotSameAlignmentReason** - 返回指定栅格中指定位置的邻域。如果指定位置为 NODATA 则返回 NODATA。
- **ST_NumBands** - 返回指定栅格中的波段数。
- **ST_Overlaps** - 返回指定栅格 A 和栅格 B 是否重叠。如果指定位置为 NODATA 则返回 NODATA。
- **ST_PixelAsCentroid** - 返回指定栅格中指定位置的质心 (centroid)。
- **ST_PixelAsCentroids** - 返回指定栅格中指定位置的质心 (centroid) 的 X, Y 坐标。
- **ST_PixelAsPoint** - 返回指定栅格中指定位置的点。
- **ST_PixelAsPoints** - 返回指定栅格中指定位置的点。返回 X, Y 坐标。
- **ST_PixelAsPolygon** - 返回指定栅格中指定位置的点。
- **ST_PixelAsPolygons** - 返回指定栅格中指定位置的点。返回 X, Y 坐标。
- **ST_PixelHeight** - 返回指定栅格中指定位置的高度。
- **ST_PixelOfValue** - 返回指定栅格中指定位置的 columnx, rowy 坐标。
- **ST_PixelWidth** - 返回指定栅格中指定位置的宽度。
- **ST_Polygon** - NODATA 返回指定栅格中指定位置的点。
- **ST_Quantile** - 返回指定栅格中指定位置的 (population) 的 quantile (quantile) 值。返回 25%, 50%, 75% (percentile) 值。
- **ST_RastFromHexWKB** - Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.
- **ST_RastFromWKB** - Return a raster value from a Well-Known Binary (WKB) raster.
- **ST_RasterToWorldCoord** - 返回指定栅格中指定位置的 X, Y (x, y) 坐标。返回 1 个值。
- **ST_RasterToWorldCoordX** - 返回指定栅格中指定位置的 X 坐标。返回 1 个值。
- **ST_RasterToWorldCoordY** - 返回指定栅格中指定位置的 Y 坐标。返回 1 个值。
- **ST_Reclass** - 返回指定栅格中指定位置的点。nband 指定波段数。nband 指定波段数。返回 1 个值。返回 16BUI 或 8BUI 值。
- **ST_ReclassExact** - Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.
- **ST_Resample** - 返回指定栅格中指定位置的点。返回 1 个值。

- **ST_Rescale** - Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline, Lanczos, Max or Min resampling algorithm. Default is NearestNeighbor.
- **ST_Resize** - 调整栅格的尺寸/分辨率。
- **ST_Reskew** - 调整 (栅格的 skew) 参数。NearestNeighbor(最近邻), Bilinear, Cubic, CubicSpline 或 Lanczos 重采样算法。最近邻 NearestNeighbor 默认。
- **ST_Rotation** - 栅格的旋转角度。
- **ST_Roughness** - DEM 栅格的“粗糙度”参数。
- **ST_SRID** - spatial_ref_sys 表中的 SRID 值。
- **ST_SameAlignment** - 检查两个栅格是否具有相同的对齐方式 (是否都是上行或下行对齐)。
- **ST_ScaleX** - 栅格的 X 轴缩放比例。
- **ST_ScaleY** - 栅格的 Y 轴缩放比例。
- **ST_SetBandIndex** - Update the external band number of an out-db band
- **ST_SetBandIsNoData** - 设置 isnodata 标志。
- **ST_SetBandNoDataValue** - NODATA 值。栅格中的 NODATA 值默认为 1。设置 NODATA 值, nodata value = NULL。
- **ST_SetBandPath** - Update the external path and band number of an out-db band
- **ST_SetGeoReference** - 设置栅格的地理参考信息。GDAL 的 ESRI 格式。
- **ST_SetM** - Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the M dimension using the requested resample algorithm.
- **ST_SetRotation** - 设置栅格的旋转角度。
- **ST_SetSRID** - 设置 SRID 值。
- **ST_SetScale** - X 和 Y 轴的缩放比例。
- **ST_SetSkew** - 设置 X 和 Y 轴的 skew (倾斜) 参数。
- **ST_SetUpperLeft** - Sets the value of the upper left corner of the pixel of the raster to projected X and Y coordinates.
- **ST_SetValue** - 设置 columnx, rowy 位置的值。
- **ST_SetValues** - 设置栅格中的值。
- **ST_SetZ** - Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimension using the requested resample algorithm.
- **ST_SkewX** - 设置 X 轴的 skew (倾斜) 参数。
- **ST_SkewY** - 设置 Y 轴的 skew (倾斜) 参数。
- **ST_Slope** - 计算栅格的坡度 (斜率)。

- **ST_SnapToGrid** - 将栅格数据 snap 到指定的栅格。NearestNeighbor(最近邻), Bilinear, Cubic, CubicSpline 或 Lanczos 插值方法。最近邻 NearestNeighbor 是默认值。
- **ST_Summary** - 返回栅格带的统计信息。
- **ST_SummaryStats** - Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed if no band is specified.
- **ST_SummaryStatsAgg** - Aggregate. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a set of raster. Band 1 is assumed if no band is specified.
- **ST_TPI** - 地形位置指数 (Topographic Position Index) 函数。
- **ST_TRI** - 地形崎岖度指数 (Terrain Ruggedness Index) 函数。
- **ST_Tile** - 将栅格数据分割成指定大小的瓦片。
- **ST_Touches** - 检查两个栅格是否接触。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。NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos 插值方法。最近邻 NearestNeighbor 是默认值。
- **ST_Union** - 将两个栅格合并为一个栅格。
- **ST_UpperLeftX** - 返回栅格左上角的 X 坐标。
- **ST_UpperLeftY** - 返回栅格左上角的 Y 坐标。
- **ST_Value** - 返回指定列和行中的值。如果指定了 exclude_nodata_value，则返回非空值。如果指定了 nodata，则返回空值。exclude_nodata_value 是默认值。
- **ST_ValueCount** - 返回栅格中每个值的计数。返回一个数组，其中每个元素是一个包含值和计数的元组。NODATA 值返回空值。
- **ST_Width** - 返回栅格的宽度。
- **ST_Within** - 检查一个栅格是否完全包含在另一个栅格中。
- **ST_WorldToRasterCoord** - 将世界坐标 (X, Y) 转换为栅格坐标 (xw, yw)。
- **ST_WorldToRasterCoordX** - 将世界坐标 (pt) 转换为栅格坐标 (xw, yw) 中的 X 坐标。
- **ST_WorldToRasterCoordY** - 将世界坐标 (pt) 转换为栅格坐标 (xw, yw) 中的 Y 坐标。
- **UpdateRasterSRID** - 更新栅格的 SRID。

13.6 PostGIS Geometry / Geography / Raster Dump Functions

The functions given below are PostGIS functions that take as input or return as output a set of or single **geometry_dump** or **geomval** data type object.

- **ST_DumpAsPolygons** - 将 **geomval(geom, val)** 转换为 **geomval**。返回一个 **geometry_dump** 类型的集合。
- **ST_Intersection** - 返回两个 **geometry** 的交集，返回一个 **geometry**。
- **ST_Dump** - Returns a set of **geometry_dump** rows for the components of a geometry.
- **ST_DumpPoints** - 返回一个 **geometry_dump** 类型的集合。
- **ST_DumpRings** - Returns a set of **geometry_dump** rows for the exterior and interior rings of a Polygon.
- **ST_DumpSegments** - 返回一个 **geometry_dump** 类型的集合。

13.7 PostGIS Box Functions

The functions given below are PostGIS functions that take as input or return as output the box* family of PostGIS spatial types. The box family of types consists of **box2d**, and **box3d**

- **Box2D** - Returns a BOX2D representing the 2D extent of a geometry.
- **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
- **MakeTopologyPrecise** - Snap topology vertices to precision grid.
- **Box3D** - 返回一个 BOX3D 类型的集合。
- **ST_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST_3DMakeBox** - Creates a BOX3D defined by two 3D point geometries.
- **ST_AsMVTGeom** - Transforms a geometry into the coordinate space of a MVT tile.
- **ST_AsTWKB** - 返回 TWKB(Tiny Well-Known Binary) 格式的数据。
- **ST_Box2dFromGeoHash** - GeoHash 转换为 BOX2D 类型。
- **ST_ClipByBox2D** - Computes the portion of a geometry falling within a rectangle.
- **ST_EstimatedExtent** - Returns the estimated extent of a spatial table.
- **ST_Expand** - Returns a bounding box expanded from another bounding box or a geometry.
- **ST_Extent** - Aggregate function that returns the bounding box of geometries.
- **ST_MakeBox2D** - Creates a BOX2D defined by two 2D point geometries.
- **ST_RemoveIrrelevantPointsForView** - Removes points that are irrelevant for rendering a specific rectangular view of a geometry.
- **ST_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- **ST_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.

- **ST_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- **ST_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **ST_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- **RemoveUnusedPrimitives** - Removes topology primitives which not needed to define existing Topo-Geometry objects.
- **ValidateTopology** - Returns a set of validate_topology_return_type 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_ApproximateMedialAxis** - 返回几何体的近似 medial axis.
- **CG_ConstrainedDelaunayTriangles** - Return a constrained Delaunay triangulation around the given input geometry.
- **CG_Extrude** - 将几何体沿 Z 轴拉伸指定的距离。
- **CG_ForceLHR** - LHR(Left Hand Reverse; 左手规则) 强制几何体为左手规则。
- **CG_IsPlanar** - 检查几何体是否是平面。
- **CG_IsSolid** - 检查几何体是否是体。如果几何体是体，则返回 true，否则返回 false。
- **CG_MakeSolid** - 将几何体转换为体。如果几何体已经是体，则返回原几何体。否则，将几何体转换为体并返回。
- **CG_Orientation** - 返回几何体的朝向 (orientation)。
- **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** - 返回几何体的 (straight skeleton)。
- **CG_Tessellate** - 将几何体转换为三角网 (tessellation) 并返回 TIN 列表。
- **CG_Visibility** - Compute a visibility polygon from a point or a segment in a polygon geometry
- **CG_Volume** - 3 维几何体的体积。如果几何体是体，则返回其体积。否则，返回 0。
- **DropGeometryColumn** - 删除几何体列。
- **GeometryType** - ST_Geometry 类型的几何体。
- **ST_3DArea** - 3 维几何体的面积。如果几何体是体，则返回其面积。否则，返回 0。
- **ST_3DClosestPoint** - g2 中的点 g1 最近的 3 维点。
- **ST_3DConvexHull** - 返回几何体的 3D 凸包。
- **ST_3DDFullyWithin** - Tests if two 3D geometries are entirely within a given 3D distance
- **ST_3DDWithin** - Tests if two 3D geometries are within a given 3D distance
- **ST_3DDifference** - 3 维几何体的差集。
- **ST_3DDistance** - 返回两个几何体之间的 3D 距离 (SRS 距离)。
- **ST_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST_3DIntersection** - 3 维几何体的交集。
- **ST_3DIntersects** - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)

- **ST_3DLength** - 返回3D几何体的长度。
- **ST_3DLineInterpolatePoint** - Returns a point interpolated along a 3D line at a fractional location.
- **ST_3DLongestLine** - 返回3D几何体中3条最长的线。
- **ST_3DMaxDistance** - 返回3D几何体中, 最大距离 (SRS 坐标系) 3 条最长的线。
- **ST_3DPerimeter** - 返回3D几何体的周长。
- **ST_3DShortestLine** - 返回3D几何体中3条最短的线。
- **ST_3DUnion** - Perform 3D union.
- **ST_AddMeasure** - Interpolates measures along a linear geometry.
- **ST_AddPoint** - 返回3D几何体添加点后的结果。
- **ST_Affine** - Apply a 3D affine transformation to a geometry.
- **ST_ApproximateMedialAxis** - 返回3D几何体的近似中轴。
- **ST_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST_AsEWKB** - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- **ST_AsEWKT** - 返回 WKT(Well-Known Text) 格式的 SRID 几何体。
- **ST_AsGML** - 返回 GML 2 或 GML 3 格式的几何体。
- **ST_AsGeoJSON** - Return a geometry or feature in GeoJSON format.
- **ST_AsHEXEWKB** - 返回 (NDR) 或 (XDR) 格式的 HEXEWKB (二进制) 几何体。
- **ST_AsKML** - 返回 GML 2 或 GML 3 格式的 KML 几何体。
- **ST_AsX3D** - 返回 X3D XML 格式的几何体: ISO-IEC-19776-1.2-X3DEncodings-XML 格式。
- **ST_Boundary** - 返回3D几何体的边界。
- **ST_BoundingDiagonal** - 返回3D几何体的包围盒的对角线。
- **ST_CPAWithin** - Tests if the closest point of approach of two trajectories is within the specified distance.
- **ST_ChaikinSmoothing** - Returns a smoothed version of a geometry, using the Chaikin algorithm
- **ST_ClosestPointOfApproach** - Returns a measure at the closest point of approach of two trajectories.
- **ST_Collect** - Creates a GeometryCollection or Multi* geometry from a set of geometries.
- **ST_ConstrainedDelaunayTriangles** - Return a constrained Delaunay triangulation around the given input geometry.
- **ST_ConvexHull** - Computes the convex hull of a geometry.
- **ST_CoordDim** - ST_Geometry 的坐标维度。
- **ST_CurveN** - Returns the Nth component curve geometry of a CompoundCurve.
- **ST_CurveToLine** - Converts a geometry containing curves to a linear geometry.
- **ST_DelaunayTriangles** - Returns the Delaunay triangulation of the vertices of a geometry.

- **ST_Difference** - Computes a geometry representing the part of geometry A that does not intersect geometry B.
- **ST_DistanceCPA** - Returns the distance between the closest point of approach of two trajectories.
- **ST_Dump** - Returns a set of geometry_dump rows for the components of a geometry.
- **ST_DumpPoints** - 返回几何体中所有点的集合。
- **ST_DumpRings** - Returns a set of geometry_dump rows for the exterior and interior rings of a Polygon.
- **ST_DumpSegments** - 返回几何体中所有线段的集合。
- **ST_EndPoint** - ST_LineString 或 ST_CircularString 的最后一个点。
- **ST_ExteriorRing** - 返回多边形的外环。
- **ST_Extrude** - 将 2D 几何体沿 Z 轴拉伸。
- **ST_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
- **ST_Force2D** - 强制将几何体转换为 2D。
- **ST_ForceCurve** - 强制将几何体转换为曲线 (upcast)。
- **ST_ForceLHR** - LHR(Left Hand Reverse; 左手规则) 强制将几何体转换为左手规则。
- **ST_ForcePolygonCCW** - Orients all exterior rings counter-clockwise and all interior rings clockwise.
- **ST_ForcePolygonCW** - Orients all exterior rings clockwise and all interior rings counter-clockwise.
- **ST_ForceRHR** - 强制将几何体转换为右手规则 (orientation) (Right-Hand Rule)。
- **ST_ForceSFS** - 强制将几何体转换为 SFS 1.1 格式。
- **ST_Force3D** - 强制将几何体转换为 3D XYZ 格式。
- **ST_Force3DZ** - 强制将几何体转换为 3D XYZ 格式，指定 Z 值。
- **ST_Force4D** - 强制将几何体转换为 4D XYZM 格式。
- **ST_ForceCollection** - 强制将几何体转换为集合。
- **ST_GeomFromEWKB** - EWKB(Extended Well-Known Binary) 格式转换为 ST_Geometry。
- **ST_GeomFromEWKT** - EWKT(Extended Well-Known Text) 格式转换为 ST_Geometry。
- **ST_GeomFromGML** - GML 格式转换为 PostGIS 几何体。
- **ST_GeomFromGeoJSON** - GeoJSON 格式转换为 PostGIS 几何体。
- **ST_GeomFromKML** - KML 格式转换为 PostGIS 几何体。
- **ST_GeometricMedian** - 计算几何体的几何中位数 (median)。
- **ST_GeometryN** - 从 ST_Geometry 集合中获取第 N 个几何体。
- **ST_GeometryType** - 返回 ST_Geometry 的几何类型。
- **ST_HasArc** - Tests if a geometry contains a circular arc
- **ST_HasM** - Checks if a geometry has an M (measure) dimension.
- **ST_HasZ** - Checks if a geometry has a Z dimension.

- **ST_InteriorRingN** - 返回几何体的第 N 个内部环。
- **ST_InterpolatePoint** - 返回沿几何体 (M 值) 的指定位置插值点。
- **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。POLYGON (多边形) 是否是 TRUE 或 FALSE。
- **ST_IsCollection** - 是否是集合, 点, 线或面 TRUE 或 FALSE。
- **ST_IsPlanar** - 是否是平面。
- **ST_IsPolygonCCW** - Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.
- **ST_IsPolygonCW** - Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.
- **ST_IsSimple** - 是否是简单的 TRUE 或 FALSE。
- **ST_IsSolid** - 是否是实体的。是否是实体的。
- **ST_IsValidTrajectory** - Tests if the geometry is a valid trajectory.
- **ST_LengthSpheroid** - 返回球面长度。
- **ST_LineFromMultiPoint** - 从多点创建线。
- **ST_LineInterpolatePoint** - Returns a point interpolated along a line at a fractional location.
- **ST_LineInterpolatePoints** - Returns points interpolated along a line at a fractional interval.
- **ST_LineSubstring** - Returns the part of a line between two fractional locations.
- **ST_LineToCurve** - Converts a linear geometry to a curved geometry.
- **ST_LocateBetweenElevations** - Returns the portions of a geometry that lie in an elevation (Z) range.
- **ST_M** - Returns the M coordinate of a Point.
- **ST_MakeLine** - 从点创建线。
- **ST_MakePoint** - Creates a 2D, 3DZ or 4D Point.
- **ST_MakePolygon** - Creates a Polygon from a shell and optional list of holes.
- **ST_MakeSolid** - 从面创建实体。从面创建实体。从面创建实体, 从面创建实体 TIN 面。
- **ST_MakeValid** - Attempts to make an invalid geometry valid without losing vertices.
- **ST_MemSize** - ST_Geometry 的内存大小。
- **ST_MemUnion** - Aggregate function which unions geometries in a memory-efficient but slower way.
- **ST_NDims** - ST_Geometry 的维度。
- **ST_NPoints** - 返回几何体 (点) 的点数。
- **ST_NRings** - 返回几何体的环数。
- **ST_Node** - Nodes a collection of lines.
- **ST_NumCurves** - Return the number of component curves in a CompoundCurve.
- **ST_NumGeometries** - 返回几何体集合中的几何体数量。

- **ST_NumPatches** - 返回几何体中的补丁数量。如果几何体为 NULL，则返回 NULL。
- **ST_Orientation** - 返回几何体的方位 (orientation)。
- **ST_PatchN** - 返回几何体的第 N 个补丁。
- **ST_PointFromWKB** - 从 SRID 和 WKB 数据创建点。
- **ST_PointN** - 返回 LineString 或 CircularString 中的第 N 个点。
- **ST_PointOnSurface** - 计算一个点，保证位于多边形或几何体内。
- **ST_Points** - 返回几何体中的所有点。
- **ST_Polygon** - 从 LineString 创建多边形，并指定 SRID。
- **ST_RemovePoint** - 从线串中移除一个点。
- **ST_RemoveRepeatedPoints** - 返回去除重复点的几何体版本。
- **ST_Reverse** - 反转几何体的方向。
- **ST_Rotate** - 围绕原点旋转几何体。
- **ST_RotateX** - 围绕 X 轴旋转几何体。
- **ST_RotateY** - 围绕 Y 轴旋转几何体。
- **ST_RotateZ** - 围绕 Z 轴旋转几何体。
- **ST_Scale** - 按给定因子缩放几何体。
- **ST_Scroll** - 更改封闭 LineString 的起始点。
- **ST_SetPoint** - 设置 LineString 中的点。
- **ST_ShiftLongitude** - 将几何体的经度坐标在 -180..180 和 0..360 之间转换。
- **ST_SnapToGrid** - 将几何体 snap 到指定的栅格。
- **ST_StartPoint** - 返回 LineString 的第一个点。
- **ST_StraightSkeleton** - 返回几何体的直线骨架。
- **ST_SwapOrdinates** - 交换几何体中每对连续点的 X 和 Y 坐标。
- **ST_SymDifference** - 计算两个几何体 A 和 B 的对称差集，即它们不重叠的部分。
- **ST_Tessellate** - 将多边形面 tessellate 成三角形 (TIN)。
- **ST_TransScale** - 按给定偏移量和因子缩放并平移几何体。
- **ST_Translate** - 按给定偏移量平移几何体。
- **ST_UnaryUnion** - 计算单个几何体所有组件的并集。
- **ST_Union** - 计算两个或多个几何体的并集。
- **ST_Volume** - 计算 3D 几何体的体积。如果几何体是 2D 的，则返回 0。
- **ST_WrapX** - 将 X 坐标包裹到指定的范围内。
- **ST_X** - 返回点的 X 坐标。
- **ST_XMax** - 返回 2D 或 3D 边界框或几何体的 X 最大值。

- **ST_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST_Y** - Returns the Y coordinate of a Point.
- **ST_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- **ST_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- **ST_Z** - Returns the Z coordinate of a Point.
- **ST_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **ST_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- **ST_Zmflag** - ST_Geometry 返回一个布尔值。
- **Equals** - 返回 TopoGeometry 是否相等。
- **Intersects** - 返回 TopoGeometry 是否相交。
- **UpdateGeometrySRID** - Updates the SRID of all features in a geometry column, and the table meta-data.
- **&&&** - A 是否包含 B 是否包含 TRUE 返回。
- **&&&(geometry,gidx)** - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- **&&&(gidx,geometry)** - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- **&&&(gidx,gidx)** - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

13.9 PostGIS Curved Geometry Support Functions

The functions given below are PostGIS functions that can use CIRCULARSTRING, CURVEPOLYGON, and other curved geometry types

- **AddGeometryColumn** - 添加几何列。
- **Box2D** - Returns a BOX2D representing the 2D extent of a geometry.
- **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
- **DropGeometryColumn** - 删除几何列。
- **GeometryType** - ST_Geometry 返回类型。
- **PostGIS_AddBBox** - 添加边界框。
- **PostGIS_DropBBox** - 删除边界框。
- **PostGIS_HasBBox** - 检查是否有边界框。
- **ST_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST_Affine** - Apply a 3D affine transformation to a geometry.
- **ST_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST_AsEWKB** - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.

- **ST_AsEWKT** - 将 WKT(Well-Known Text) 转换为 SRID 指定的几何类型。
- **ST_AsHEXEWKB** - 将 (NDR) 或 (XDR) 格式的 HEXEWKB (二进制) 转换为文本格式。
- **ST_AsSVG** - Returns SVG path data for a geometry.
- **ST_AsText** - 将 WKT(Well-Known Text) 转换为 SRID 指定的几何类型。
- **ST_ClusterDBSCAN** - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- **ST_ClusterWithin** - Aggregate function that clusters geometries by separation distance.
- **ST_ClusterWithinWin** - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- **ST_Collect** - Creates a GeometryCollection or Multi* geometry from a set of geometries.
- **ST_CoordDim** - ST_Geometry 的坐标维度。
- **ST_CurveToLine** - Converts a geometry containing curves to a linear geometry.
- **ST_Distance** - 返回 3 个 (最长) 距离。
- **ST_Dump** - Returns a set of geometry_dump rows for the components of a geometry.
- **ST_DumpPoints** - 将几何体分解为点。
- **ST_EndPoint** - ST_LineString 或 ST_CircularString 的终点。
- **ST_EstimatedExtent** - Returns the estimated extent of a spatial table.
- **ST_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
- **ST_Force2D** - 将 "2 维" 强制为 2D。
- **ST_ForceCurve** - 将几何体强制为曲线 (upcast)。
- **ST_ForceSFS** - 将 SFS 1.1 强制为 SFS 1.1。
- **ST_Force3D** - 将 XYZ 强制为 3D。ST_Force3DZ 强制为 3DZ。
- **ST_Force3DM** - 将 XYM 强制为 3DM。
- **ST_Force3DZ** - 将 XYZ 强制为 3DZ。
- **ST_Force4D** - 将 XYZM 强制为 4D。
- **ST_ForceCollection** - 将几何体强制为集合。
- **ST_GeoHash** - 将几何体转换为 GeoHash。
- **ST_GeogFromWKB** - WKB 转换为 EWKB(或 WKB) 的地理几何。
- **ST_GeomFromEWKB** - EWKB(Extended Well-Known Binary) 转换为 ST_Geometry 几何。
- **ST_GeomFromEWKT** - EWKT(Extended Well-Known Text) 转换为 ST_Geometry 几何。
- **ST_GeomFromText** - WKT 转换为 ST_Geometry 几何。
- **ST_GeomFromWKB** - WKB(Well-Known Binary) 转换为 SRID 指定的几何类型。
- **ST_GeometryN** - ST_Geometry 的第 N 个几何体。

- **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **&<|** - A 与 B 相交 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 是否闭合 TRUE. 多边形 (多边形) 是否 TRUE.
- **ST_IsCollection** - 是否是集合, 点, 面 TRUE.
- **ST_IsEmpty** - Tests if a geometry is empty.
- **ST_LineToCurve** - Converts a linear geometry to a curved geometry.
- **ST_MemSize** - ST_Geometry 内存大小.
- **ST_NPoints** - 几何体 (点) 数量.
- **ST_NRings** - 多边形环数量.
- **ST_PointFromWKB** - 从 SRID 的 WKB 格式转换.
- **ST_PointN** - ST_LineString 或 ST_CircularString 的第 N 个点.
- **ST_Points** - 几何体所有点.
- **ST_Rotate** - Rotates a geometry about an origin point.
- **ST_RotateZ** - Rotates a geometry about the Z axis.
- **ST_SRID** - Returns the spatial reference identifier for a geometry.
- **ST_Scale** - Scales a geometry by given factors.
- **ST_SetSRID** - Set the SRID on a geometry.
- **ST_StartPoint** - Returns the first point of a LineString.
- **ST_Summary** - 几何体摘要.
- **ST_SwapOrdinates** - 交换几何体的经纬度.
- **ST_TransScale** - Translates and scales a geometry by given offsets and factors.
- **ST_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST_Translate** - Translates a geometry by given offsets.
- **ST_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- **ST_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- **ST_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- **ST_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **ST_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- **ST_Zmflag** - ST_Geometry 是否 3D.

- **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 bounding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box contains a 2D float precision bounding box (BOX2DF).
- **&&** - A 2D BOX2DF B 2D BOX2DF TRUE.
- **&&&** - A n-D BOX2DF B n-D 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** - 3D area. Returns 0 if not a surface.
- **CG_3DConvexHull** - 3D convex hull.
- **CG_3DDifference** - 3D difference.
- **CG_3DIntersection** - 3D intersection.
- **CG_3DUnion** - Perform 3D union using postgis_sfcgal.

- **CG_ApproximateMedialAxis** - 计算多边形的近似 medial axis.
- **CG_Extrude** - 将 2D 几何体沿指定方向拉伸.
- **CG_ForceLHR** - LHR(Left Hand Reverse; 左手规则) 强制使用.
- **CG_IsPlanar** - 检查几何体是否共面.
- **CG_IsSolid** - 检查几何体是否为有效的体素. 返回 true 或 false.
- **CG_MakeSolid** - 将 2D 几何体转换为体素. 返回体素 ID, 面 ID, 边 ID, 点 ID, 以及 TIN 面 ID.
- **CG_StraightSkeleton** - 计算几何体的 straight skeleton.
- **CG_Tessellate** - 对几何体进行 tessellation. 返回 TIN 面 ID.
- **CG_Visibility** - Compute a visibility polygon from a point or a segment in a polygon geometry
- **CG_Volume** - 3D 几何体的体积. 返回浮点数值 (0 到无穷大).
- **GeometryType** - ST_Geometry 类型的函数.
- **ST_3DArea** - 3D 几何体的面积. 返回浮点数值 (0 到无穷大).
- **ST_3DClosestPoint** - g2 中的点 g1 的最近点. 返回 3D 点 ID.
- **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** - 两个 3D 几何体 (SRS 相同) 之间的 3D 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** - 3D 几何体的最长线 (longest) ID.
- **ST_3DMaxDistance** - 两个 3D 几何体 (SRS 相同) 之间的最大 3D 距离.
- **ST_3DShortestLine** - 3D 几何体的最短线 (shortest) ID.
- **ST_3DUnion** - Perform 3D union.
- **ST_Affine** - Apply a 3D affine transformation to a geometry.
- **ST_ApproximateMedialAxis** - 计算多边形的近似 medial axis.
- **ST_Area** - 几何体的面积.
- **ST_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST_AsEWKB** - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.

- **ST_AsEWKT** - 将 WKT(Well-Known Text) 转换为 SRID 指定的坐标系。
- **ST_AsGML** - 将 GML 2 或 GML 3 转换为 XML。
- **ST_AsX3D** - 将 X3D XML 转换为 ISO-IEC-19776-1.2-X3DEncodings-XML 格式。
- **ST_CoordDim** - ST_Geometry 的坐标维度。
- **ST_Dimension** - ST_Geometry 的维度。
- **ST_Dump** - Returns a set of geometry_dump rows for the components of a geometry.
- **ST_DumpPoints** - 将几何体分解为点。
- **ST_Expand** - Returns a bounding box expanded from another bounding box or a geometry.
- **ST_Extent** - Aggregate function that returns the bounding box of geometries.
- **ST_Extrude** - 将几何体沿 Z 轴拉伸。
- **ST_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
- **ST_Force2D** - 将几何体强制为 2D。
- **ST_ForceLHR** - LHR(Left Hand Reverse; 左手规则) 强制几何体。
- **ST_ForceRHR** - RHR(Right Hand Rule; 右手规则) 强制几何体 (orientation) 强制几何体 (Right-Hand Rule) 强制几何体。
- **ST_ForceSFS** - 将 SFS 1.1 强制几何体。
- **ST_Force3D** - 将 XYZ 强制几何体。ST_Force3DZ 强制几何体。
- **ST_Force3DZ** - 将 XYZ 强制几何体。
- **ST_ForceCollection** - 将几何体强制为集合。
- **ST_GeomFromEWKB** - EWKB(Extended Well-Known Binary) 强制 ST_Geometry 几何体。
- **ST_GeomFromEWKT** - EWKT(Extended Well-Known Text) 强制 ST_Geometry 几何体。
- **ST_GeomFromGML** - 将 GML 强制为 PostGIS 几何体。
- **ST_GeometryN** - ST_Geometry 的第 N 个几何体。
- **ST_GeometryType** - ST_Geometry 的几何类型。
- **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **<|** - A 与 B 不相交 TRUE。
- **~=** - A 与 B 不相交 TRUE。
- **ST_IsClosed** - LINESTRING 是否闭合 TRUE。POLYGON (多边形) 是否 TRUE。
- **ST_IsPlanar** - 几何体是否平面。
- **ST_IsSolid** - 几何体是否实心。是否实心。
- **ST_MakeSolid** - 将几何体强制为实心。将几何体强制为实心。将几何体强制为实心, 将几何体强制为实心 TIN 强制几何体。
- **ST_MemSize** - ST_Geometry 的内存大小。
- **ST_NPoints** - 几何体 (多边形) 的点数。

- **ST_NumGeometries** - 返回几何对象中的几何数量。返回几何对象中的几何数量。
- **ST_NumPatches** - 返回几何对象中的补丁数量。如果几何对象为 NULL 则返回 0。
- **ST_PatchN** - ST_Geometry 返回几何对象中的第 N 个补丁。
- **ST_RemoveRepeatedPoints** - Returns a version of a geometry with duplicate points removed.
- **ST_Reverse** - 返回几何对象的反向几何。
- **ST_Rotate** - Rotates a geometry about an origin point.
- **ST_RotateX** - Rotates a geometry about the X axis.
- **ST_RotateY** - Rotates a geometry about the Y axis.
- **ST_RotateZ** - Rotates a geometry about the Z axis.
- **ST_Scale** - Scales a geometry by given factors.
- **ST_ShiftLongitude** - Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- **ST_StraightSkeleton** - 返回几何对象的 (straight skeleton) 骨架。
- **ST_Summary** - 返回几何对象的摘要。
- **ST_SwapOrdinates** - 交换几何对象的 X 和 Y 坐标。
- **ST_Tessellate** - 返回几何对象的 (tessellation) 三角网。返回 TIN 或 TIN 列表。
- **ST_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST_Volume** - 3 维几何对象的体积。返回几何对象的体积 (如果为 0 则返回 0)。
- **~(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bounding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box contains a 2D float precision bounding box (BOX2DF).
- **&&** - A 2D 几何对象 B 2D 几何对象 TRUE 如果相交。
- **&&&** - A n 维几何对象 B n 维几何对象 TRUE 如果相交。
- **@(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- **@(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- **@(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- **&&(box2df,box2df)** - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- **&&(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- **&&(geometry,box2df)** - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

- **&&&(geometry,gidx)** - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- **&&&(gidx,geometry)** - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- **&&&(gidx,gidx)** - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

13.11 PostGIS Function Support Matrix

Below is an alphabetical listing of spatial specific functions in PostGIS and the kinds of spatial types they work with or OGC/SQL compliance they try to conform to.

- A  means the function works with the type or subtype natively.
- A  means it works but with a transform cast built-in using cast to geometry, transform to a "best srid" spatial ref and then cast back. Results may not be as expected for large areas or areas at poles and may accumulate floating point junk.
- A  means the function works with the type because of a auto-cast to another such as to box3d rather than direct type support.
- A  means the function only available if PostGIS compiled with SFCGAL support.
- geom - Basic 2D geometry support (x,y).
- geog - Basic 2D geography support (x,y).
- 2.5D - basic 2D geometries in 3 D/4D space (has Z or M coord).
- PS - Polyhedral surfaces
- T - Triangles and Triangulated Irregular Network surfaces (TIN)

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Collect	✓		✓	✓			
ST_LineFromMultiPoint	✓		✓				
ST_MakeEnvelope	✓						
ST_MakeLine	✓		✓				
ST_MakePoint	✓		✓				
ST_MakePointM	✓						
ST_MakePolygon	✓		✓				
ST_Point	✓				✓		
ST_PointZ	✓						
ST_PointM	✓						
ST_PointZM	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Polygon	✓		✓		✓		
ST_TileEnvelope	✓						
ST_HexagonGrid	✓						
ST_Hexagon	✓						
ST_SquareGrid	✓						
ST_Square	✓						
ST_Letters	✓						
GeometryType	✓		✓	✓		✓	✓
ST_Boundary	✓		✓		✓		
ST_BoundingDiagonal	✓		✓				
ST_CoordDim	✓		✓	✓	✓	✓	✓
ST_Dimension	✓				✓	✓	✓
ST_Dump	✓		✓	✓		✓	✓
ST_DumpPoints	✓		✓	✓		✓	✓
ST_DumpSegments	✓		✓				✓
ST_DumpRings	✓		✓				
ST_EndPoint	✓		✓	✓	✓		
ST_Envelope	✓				✓		
ST_ExteriorRing	✓		✓		✓		
ST_GeometryN	✓		✓	✓	✓	✓	✓
ST_GeometryType	✓		✓		✓	✓	
ST_HasArc	✓		✓	✓			
ST_InteriorRing	✓		✓		✓		
ST_NumCurves	✓		✓		✓		
ST_CurveN	✓		✓		✓		
ST_IsClosed	✓		✓	✓	✓	✓	
ST_IsCollection	✓		✓	✓			
ST_IsEmpty	✓			✓	✓		
ST_IsPolygonCCW	✓		✓				
ST_IsPolygonCW	✓		✓				
ST_IsRing	✓				✓		
ST_IsSimple	✓		✓		✓		
ST_M	✓		✓		✓		
ST_MemSize	✓		✓	✓		✓	✓

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_NDims	✓		✓				
ST_NPoints	✓		✓	✓		✓	
ST_NRings	✓		✓	✓			
ST_NumGeometries	✓		✓		✓	✓	✓
ST_NumInteriorRings	✓				✓		
ST_NumInteriorRing	✓						
ST_NumPatches	✓		✓		✓	✓	
ST_NumPoints	✓				✓		
ST_PatchN	✓		✓		✓	✓	
ST_PointN	✓		✓	✓	✓		
ST_Points	✓		✓	✓			
ST_StartPoint	✓		✓	✓	✓		
ST_Summary	✓	✓		✓		✓	✓
ST_X	✓		✓		✓		
ST_Y	✓		✓		✓		
ST_Z	✓		✓		✓		
ST_Zmflag	✓		✓	✓			
ST_HasZ	✓		✓				
ST_HasM	✓		✓				
ST_AddPoint	✓		✓				
ST_CollectionExtract	✓						
ST_CollectionHomogenize	✓						
ST_CurveToLine	✓		✓	✓	✓		
ST_Scroll	✓		✓				
ST_FlipCoordinates	✓		✓	✓		✓	✓
ST_Force2D	✓		✓	✓		✓	
ST_Force3D	✓		✓	✓		✓	
ST_Force3DZ	✓		✓	✓		✓	
ST_Force3DM	✓			✓			
ST_Force4D	✓		✓	✓			
ST_ForceCollection	✓		✓	✓		✓	
ST_ForceCurve	✓		✓	✓			
ST_ForcePolygonCW	✓		✓				
ST_ForcePolygonCW	✓		✓				

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_ForceSFS	✓		✓	✓		✓	✓
ST_ForceRHR	✓		✓			✓	
ST_LineExtend	✓						
ST_LineToCurve	✓		✓	✓			
ST_Multi	✓						
ST_Normalize	✓						
ST_Project	✓	✓					
ST_QuantizeCoordinates	✓						
ST_RemovePoint	✓		✓				
ST_RemoveRepeatedPoints	✓		✓			✓	
ST_RemoveIrrelevantPointsForView	✓						
ST_RemoveSmallParts	✓						
ST_Reverse	✓		✓			✓	
ST_Segmentize	✓	✓					
ST_SetPoint	✓		✓				
ST_ShiftLongitude	✓		✓			✓	✓
ST_WrapX	✓		✓				
ST_SnapToGrid	✓		✓				
ST_Snap	✓						
ST_SwapOrdinate	✓		✓	✓		✓	✓
ST_IsValid	✓				✓		
ST_IsValidDetail	✓						
ST_IsValidReason	✓						
ST_MakeValid	✓		✓				
ST_InverseTransformPipeline	✓						
ST_SetSRID	✓			✓			
ST_SRID	✓			✓	✓		
ST_Transform	✓			✓	✓	✓	
ST_TransformPipeline	✓						
postgis_srs_codes							
postgis_srs							
postgis_srs_all							
postgis_srs_search	✓						
ST_BdPolyFromText	✓						
ST_BdMPolyFromText	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_GeogFromText		✓					
ST_GeographyFromText		✓					
ST_GeomCollFr	✓ Text				✓		
ST_GeomFromE	✓ CT		✓	✓		✓	✓
ST_GeomFromM	✓ RC21						
ST_GeometryFr	✓ Text				✓		
ST_GeomFromI	✓			✓	✓		
ST_LineFromTe	✓				✓		
ST_MLineFrom	✓ t				✓		
ST_MPointFrom	✓ ct				✓		
ST_MPolyFrom	✓ t				✓		
ST_PointFromTe	✓				✓		
ST_PolygonFrom	✓ ext				✓		
ST_WKTToSQL	✓				✓		
ST_GeogFromWKB		✓		✓			
ST_GeomFromE	✓ CB		✓	✓		✓	✓
ST_GeomFromV	✓ 3			✓	✓		
ST_LineFromW	✓				✓		
ST_LinestringFr	✓ WKB				✓		
ST_PointFromW	✓		✓	✓	✓		
ST_WKBToSQL	✓				✓		
ST_Box2dFromC	✓ Hash						
ST_GeomFromC	✓ Hash						
ST_GeomFromC	✓		✓			✓	✓
ST_GeomFromC	✓ JSON		✓				
ST_GeomFromk	✓		✓				
ST_GeomFromI	✓ CB						
ST_GMLToSQL	✓				✓		
ST_LineFromEn	✓ ledPolyline						
ST_PointFromGeoHash							
ST_FromFlatGeobufToTable							
ST_FromFlatGeobuf							
ST_AsEWKT	✓	✓	✓	✓		✓	✓
ST_AsText	✓	✓		✓	✓		
ST_AsBinary	✓	✓	✓	✓	✓	✓	✓

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_AsEWKB	✓		✓	✓		✓	✓
ST_AsHEXEWKB	✓		✓	✓			
ST_AsEncodedPolyline	✓						
ST_AsFlatGeobuf	✗						
ST_AsGeobuf	✗						
ST_AsGeoJSON	✓	✓	✓				
ST_AsGML	✓	✓	✓		✓	✓	✓
ST_AsKML	✓	✓	✓				
ST_AsLatLonText	✓						
ST_AsMARC21	✓						
ST_AsMVTGeom	✓						
ST_AsMVT	✗						
ST_AsSVG	✓	✓		✓			
ST_AsTWKB	✓						
ST_AsX3D	✓		✓			✓	✓
ST_GeoHash	✓			✓			
&&	✓	✓		✓		✓	
&&(geometry,box2df)	✓			✓		✓	
&&(box2df,geometry)	✓			✓		✓	
&&(box2df,box2df)	✗			✓		✓	
&&&	✓		✓	✓		✓	✓
&&&(geometry,box2df)	✓		✓	✓		✓	✓
&&&(gidx,geometry)	✓		✓	✓		✓	✓
&&&(gidx,gidx)			✓	✓		✓	✓
&<	✓						
&<	✓			✓		✓	
&>	✓						
<<	✓						
<<	✓						
=	✓	✓		✓		✓	
>>	✓						
@	✓						
@(geometry,box2df)	✓			✓		✓	
@(box2df,geometry)	✓			✓		✓	

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
@(box2df,box2df)	✓			✓		✓	
&>	✓						
>>	✓						
~	✓						
~(geometry,box2df)	✓			✓		✓	
~(box2df,geometry)	✓			✓		✓	
~(box2df,box2df)	✓			✓		✓	
~=	✓					✓	
<->	✓	✓					
=	✓						
<#>	✓						
<<->>	✓						
ST_3DIntersects	✓		✓		✓	✓	✓
ST_Contains	✓				✓		
ST_ContainsProperly	✓						
ST_CoveredBy	✓	✓					
ST_Covers	✓	✓					
ST_Crosses	✓				✓		
ST_Disjoint	✓				✓		
ST_Equals	✓				✓		
ST_Intersects	✓	✓		✓	✓		✓
ST_LineCrossingDirection	✓						
ST_OrderingEquals	✓				✓		
ST_Overlaps	✓				✓		
ST_Relate	✓				✓		
ST_RelateMatch							
ST_Touches	✓				✓		
ST_Within	✓				✓		
ST_3DDWithin	✓		✓		✓	✓	
ST_3DDFullyWithin	✓		✓			✓	
ST_DFullyWithin	✓						
ST_DWithin	✓	✓					
ST_PointInsideCircle	✓						
ST_Area	✓	✓			✓	✓	

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Azimuth	✓	✓					
ST_Angle	✓						
ST_ClosestPoint	✓	✓					
ST_3DClosestPoint	✓		✓			✓	
ST_Distance	✓	✓		✓	✓		
ST_3DDistance	✓		✓		✓	✓	
ST_DistanceSphere	✓						
ST_DistanceSphereoid	✓						
ST_FrechetDistance	✓						
ST_HausdorffDistance	✓						
ST_Length	✓	✓			✓		
ST_Length2D	✓						
ST_3DLength	✓		✓		✓		
ST_LengthSphereoid	✓		✓				
ST_LongestLine	✓						
ST_3DLongestLine	✓		✓			✓	
ST_MaxDistance	✓						
ST_3DMaxDistance	✓		✓			✓	
ST_MinimumClearance	✓						
ST_MinimumClearanceLine	✓						
ST_Perimeter	✓	✓			✓		
ST_Perimeter2D	✓						
ST_3DPerimeter	✓		✓		✓		
ST_ShortestLine	✓	✓					
ST_3DShortestLine	✓		✓			✓	
ST_ClipByBox2D	✓						
ST_Difference	✓		✓		✓		
ST_Intersection	✓	😄	✓		✓		
ST_MemUnion	✓		✓				
ST_Node	✓		✓				
ST_Split	✓						
ST_Subdivide	✓						
ST_SymDifference	✓		✓		✓		
ST_UnaryUnion	✓		✓				

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Union	✓		✓		✓		
ST_Buffer	✓	😬			✓		
ST_BuildArea	✓						
ST_Centroid	✓	✓			✓		
ST_ChaikinSmoothing	✓		✓				
ST_ConcaveHull	✓						
ST_ConvexHull	✓		✓		✓		
ST_DelaunayTriangles	✓		✓				✓
ST_FilterByM	✓						
ST_GeneratePoints	✓						
ST_GeometricMean	✓		✓				
ST_LineMerge	✓						
ST_MaximumInscribedCircle	✓						
ST_LargestEmptyCircle	✓						
ST_MinimumBoundingCircle	✓						
ST_MinimumBoundingRadius	✓						
ST_OrientedEnvelope	✓						
ST_OffsetCurve	✓						
ST_PointOnSurface	✓		✓		✓		
ST_Polygonize	✓						
ST_ReducePrecision	✓						
ST_SharedPaths	✓						
ST_Simplify	✓						
ST_SimplifyPreserveTopology	✓						
ST_SimplifyPolygonHull	✓						
ST_SimplifyVW	✓						
ST_SetEffectiveArea	✓						
ST_TriangulatePolygon	✓						
ST_VoronoiLines	✓						
ST_VoronoiPolygons	✓						
ST_CoverageInvalidEdges	✓						
ST_CoverageSimplify	✓						
ST_CoverageUnion	✓						
ST_CoverageClear	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Affine	✓		✓	✓		✓	✓
ST_Rotate	✓		✓	✓		✓	✓
ST_RotateX	✓		✓			✓	✓
ST_RotateY	✓		✓			✓	✓
ST_RotateZ	✓		✓	✓		✓	✓
ST_Scale	✓		✓	✓		✓	✓
ST_Translate	✓		✓	✓			
ST_TransScale	✓		✓	✓			
ST_ClusterDBSCAN	✓			✓			
ST_ClusterIntersecting	✓						
ST_ClusterIntersectingWin	✓						
ST_ClusterKMeans	✓						
ST_ClusterWithin	✓			✓			
ST_ClusterWithin/in	✓			✓			
Box2D	✓			✓		✓	✓
Box3D	✓		✓	✓		✓	✓
ST_EstimatedExtent	✗			✓			
ST_Expand	✓					✓	✓
ST_Extent	✓					✓	✓
ST_3DExtent	✓		✓	✓		✓	✓
ST_MakeBox2D	✓						
ST_3DMakeBox	✓						
ST_XMax	✗		✓	✓			
ST_XMin	✗		✓	✓			
ST_YMax	✗		✓	✓			
ST_YMin	✗		✓	✓			
ST_ZMax	✗		✓	✓			
ST_ZMin	✗		✓	✓			
ST_LineInterpolatePoint	✓	✓	✓				
ST_3DLineInterpolatePoint	✓		✓				
ST_LineInterpolatePoints	✓	✓	✓				
ST_LineLocatePoint	✓	✓					
ST_LineSubstring	✓	✓	✓				
ST_LocateAlong	✓				✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_LocateBetween	✓				✓		
ST_LocateBetweenElevations	✓		✓				
ST_InterpolatePoint	✓		✓				
ST_AddMeasure	✓		✓				
ST_IsValidTrajectory	✓		✓				
ST_ClosestPointApproach	✓		✓				
ST_DistanceCPA	✓		✓				
ST_CPAWithin	✓		✓				
postgis.gdal_datapath							
postgis.gdal_enabled_drivers							
postgis.enable_outdb_rasters							
postgis.gdal_vsi_options							
postgis.gdal_cpl_debug							
PostGIS_AddBB	✓			✓			
PostGIS_DropBB	✓			✓			
PostGIS_HasBB	✓			✓			
postgis_sfcgal_version							
postgis_sfcgal_full_version							
CG_ForceLHR							
CG_IsPlanar							
CG_IsSolid							
CG_MakeSolid							
CG_Orientation							
CG_Area							
CG_3DArea							
CG_Volume							
ST_ForceLHR							
ST_IsPlanar							
ST_IsSolid							
ST_MakeSolid							
ST_Orientation							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_3DArea							
ST_Volume							
CG_Intersection							
CG_Intersects							
CG_3DIntersect							
CG_Difference							
ST_3DDifference							
CG_3DDifference							
CG_Distance							
CG_3DDistance							
ST_3DConvexH							
CG_3DConvexH							
ST_3DIntersect							
CG_3DIntersect							
CG_Union							
ST_3DUnion							
CG_3DUnion							
ST_AlphaShape	 ✓						
CG_AlphaShape							
CG_ApproxConvexPartition							
ST_ApproximateMedialAxis							
CG_ApproximateMedialAxis							
ST_ConstrainedDelaunayTriangles							
CG_ConstrainedDelaunayTriangles							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Extrude							
CG_Extrude							
CG_ExtrudeStraightSkeleton							
CG_GreeneApproxConvexPartition							
ST_MinkowskiSum							
CG_MinkowskiSum							
ST_OptimalAlphaShape							
CG_OptimalAlphaShape							
CG_OptimalCorridorPartition							
CG_StraightSkeleton							
ST_StraightSkeleton							
ST_Tessellate							
CG_Tessellate							
CG_Triangulate							
CG_Visibility							
CG_YMonotonePartition							
CG_StraightSkeletonPartition							
CG_3DBuffer							
CG_Rotate							
CG_2DRotate							
CG_3DRotate							
CG_RotateX							
CG_RotateY							
CG_RotateZ							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
CG_Scale							
CG_3DScale							
CG_3DScaleArcCenter							
CG_Translate							
CG_3DTranslate							
CG_Simplify							
CG_3DAlphaWrapping							
getfaceedges_returntype							
TopoGeometry							
validatetopology_returntype							
TopoElement							
TopoElementArray							
AddTopoGeometryColumn							
RenameTopoGeometryColumn							
DropTopology							
RenameTopology							
DropTopoGeometryColumn							
FixCorruptTopoGeometryColumn							
Populate_Topology_Layer							
TopologySummary							
ValidateTopology✓							
ValidateTopologyRelation							
ValidateTopology✓recision							
MakeTopologyP✓ise							
FindTopology✓							
FindLayer✓							
TotalTopologySize							
UpgradeTopology							
CreateTopology							
CopyTopology							
ST_InitTopoGeo					✓		
ST_CreateTopoG✓					✓		
TopoGeo_AddP✓							
TopoGeo_AddLi✓string							
TopoGeo_AddP✓on							
TopoGeo_LoadC✓metry							
ST_AddIsoNode✓					✓		
ST_AddIsoEdge✓					✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_AddEdgeNewFaces	✓				✓		
ST_AddEdgeModify	✓				✓		
ST_RemEdgeNewFace					✓		
ST_RemEdgeModFace					✓		
ST_ChangeEdgeGeom	✓				✓		
ST_ModEdgeSplit	✓				✓		
ST_ModEdgeHeal					✓		
ST_NewEdgeHeal					✓		
ST_MoveIsoNode	✓				✓		
ST_NewEdgesSplit	✓				✓		
ST_RemoveIsoNode					✓		
ST_RemoveIsoEdge					✓		
GetEdgeByPoint	✓						
GetFaceByPoint	✓						
GetFaceContainingPoint	✓						
GetNodeByPoint	✓						
GetTopologyID							
GetTopologySRID							
GetTopologyName							
ST_GetFaceEdges					✓		
ST_GetFaceGeometry	✓				✓		
GetRingEdges							
GetNodeEdges							
Polygonize							
AddNode	✓						
AddEdge	✓						
AddFace	✓						
ST_Simplify	✓						
RemoveUnusedNodes	✓						
CreateTopoGeom	✓						
toTopoGeom	✓						
TopoElementArray_Agg							
TopoElement	✓						
clearTopoGeom	✓						
TopoGeom_addLineament	✓						
TopoGeom_removeLineament	✓						
TopoGeom_addToGeom	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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							
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							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Summary							
ST_UpperLeftX							
ST_UpperLeftY							
ST_Width							
ST_WorldToRas ✓	Coord						
ST_WorldToRas ✓	CoordX						
ST_WorldToRas ✓	CoordY						
ST_BandMetaData							
ST_BandNoDataValue							
ST_BandIsNoData							
ST_BandPath							
ST_BandFileSize							
ST_BandFileTimestamp							
ST_BandPixelType							
ST_MinPossibleValue							
ST_HasNoBand							
ST_PixelAsPolygons ✓							
ST_PixelAsPolygons							
ST_PixelAsPoint ✓							
ST_PixelAsPoints							
ST_PixelAsCentroids ✓	1						
ST_PixelAsCentroids							
ST_Value ✓							
ST_NearestValue ✓							
ST_SetZ ✓							
ST_SetM ✓							
ST_Neighborhood ✓							
ST_SetValue ✓							
ST_SetValues							
ST_DumpValues							
ST_PixelOfValue							
ST_SetGeoReference							
ST_SetRotation							
ST_SetScale							
ST_SetSkew							
ST_SetSRID							
ST_SetUpperLeft							
ST_Resample							
ST_Rescale							
ST_Reskew							
ST_SnapToGrid							
ST_Resize							
ST_Transform							
ST_SetBandNoDataValue							
ST_SetBandIsNoData							
ST_SetBandPath							
ST_SetBandIndex							
ST_Count							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
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							
ST_Union							
ST_Distinct4ma							
ST_InvDistWeight4ma							
ST_Max4ma							
ST_Mean4ma							
ST_Min4ma							
ST_MinDist4ma							
ST_Range4ma							
ST_StdDev4ma							
ST_Sum4ma							
ST_Aspect							
ST_HillShade							
ST_Roughness							
ST_Slope							
ST_TPI							
ST_TRI							
ST_InterpolateF	✓						
ST_Contour							
Box3D	✓						
ST_ConvexHull	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_DumpAsPolygons							
ST_Envelope	✓						
ST_MinConvexHull	✓						
ST_Polygon	✓						
ST_Intersection	✓	ctions					
&&	✓						
&<							
&>							
=							
@	✓						
~ =							
~	✓						
ST_Contains							
ST_ContainsProperly							
ST_Covers							
ST_CoveredBy							
ST_Disjoint							
ST_Intersects	✓						
ST_Overlaps							
ST_Touches							
ST_SameAlignment							
ST_NotSameAlignmentReason							
ST_Within							
ST_DWithin							
ST_DFullyWithin							
stdaddr							
rules							
table							
lex table							
gaz table							
debug_standardize_address							
parse_address							
standardize_address							
Drop_Indexes_Generate_Script							
Drop_Nation_Tables_Generate_Script							
Drop_State_Tables_Generate_Script							
Geocode	✓						
Geocode_Intersections	✓						
Get_Geocode_Setting							
Get_Tract	✓						
Install_Missing_Indexes							
Loader_Generate_Census_Script							
Loader_Generate_Script							
Loader_Generate_Nation_Script							
Missing_Indexes_Generate_Script							
Normalize_Address							
Pgc_Normalize_Address							
Pprint_Addy							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
Reverse_Geocode	✓						
Topology_Load_Tiger							
Set_Geocode_Setting							

13.12 New, Enhanced or changed PostGIS Functions

13.12.1 PostGIS Functions new or enhanced in 3.6

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.6

- **CG_2DRotate** - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry by a given angle around a specified point in 2D.
- **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_CoverageClean** - Availability: 3.6.0 - requires GEOS >= 3.14.0 Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.
- **ST_IntersectionFractions** - Availability: 3.6.0 Requires GEOS 3.14 or higher. Calculates the fraction of each raster cell that is covered by a given geometry.
- **ST_ReclassExact** - Availability: 3.6.0 Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.
- **TotalTopologySize** - Availability: 3.6.0 Total disk space used by the specified topology, including all indexes and TOAST data.
- **UpgradeTopology** - Availability: 3.6.0 Upgrades the specified topology to support large ids (int8) for topology and primitive ids.
- **ValidateTopologyPrecision** - Availability: 3.6.0 Returns non-precise vertices in the topology.
- **postgis.gdal_cpl_debug** - Availability: 3.6.0 A boolean configuration to turn logging of GDAL debug messages on and off.

13.12.2 PostGIS Functions new or enhanced in 3.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.5

- **CG_3DArea** - Availability: 3.5.0 3 参数函数。返回 3D 几何体的面积。
- **CG_3DConvexHull** - Availability: 3.5.0 1 参数函数。返回 3D 几何体的凸包。
- **CG_3DDifference** - Availability: 3.5.0 3 参数函数。返回 3D 几何体的差集。
- **CG_3DDistance** - Availability: 3.5.0 Computes the minimum 3D distance between two geometries
- **CG_3DIntersection** - Availability: 3.5.0 3 参数函数。返回 3D 几何体的交集。
- **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 1 参数函数。返回 3D 几何体的近似中轴。
- **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 2 参数函数。返回 3D 几何体的拉伸。
- **CG_ExtrudeStraightSkeleton** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Straight Skeleton Extrusion

- **CG_ForceLHR** - Availability: 3.5.0 LHR(Left Hand Reverse; 逆时针) 强制多边形。
- **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 测试多边形是否是平面的。
- **CG_IsSolid** - Availability: 3.5.0 测试多边形是否是实体的。 测试多边形是否是实体的。
- **CG_MakeSolid** - Availability: 3.5.0 测试多边形是否是实体的。 测试多边形是否是实体的。 测试多边形是否是实体的。 测试多边形是否是实体的。 TIN 测试多边形是否是实体的。
- **CG_MinkowskiSum** - Availability: 3.5.0 测试多边形是否是实体的。
- **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 测试多边形 (orientation) 测试多边形是否是实体的。
- **CG_StraightSkeleton** - Availability: 3.5.0 测试多边形是否是实体的 (straight skeleton) 测试多边形是否是实体的。
- **CG_Tessellate** - Availability: 3.5.0 测试多边形是否是实体的 (tessellation) 测试多边形是否是实体的。 TIN 测试多边形是否是实体的。
- **CG_Triangulate** - Availability: 3.5.0 Triangulates a polygonal geometry
- **CG_Union** - Availability: 3.5.0 Computes the union of two geometries
- **CG_Visibility** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Compute a visibility polygon from a point or a segment in a polygon geometry
- **CG_Volume** - Availability: 3.5.0 3 测试多边形是否是实体的。 测试多边形是否是实体的 (测试多边形是否是实体的) 0 测试多边形是否是实体的。
- **CG_YMonotonePartition** - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Computes y-monotone partition of the polygon geometry
- **ST_HasM** - Availability: 3.5.0 Checks if a geometry has an M (measure) dimension.
- **ST_HasZ** - Availability: 3.5.0 Checks if a geometry has a Z dimension.
- **ST_RemoveIrrelevantPointsForView** - Availability: 3.5.0 Removes points that are irrelevant for rendering a specific rectangular view of a geometry.
- **ST_RemoveSmallParts** - Availability: 3.5.0 Removes small parts (polygon rings or linestrings) of a geometry.
- **TopoGeo_LoadGeometry** - Availability: 3.5.0 Load a geometry into an existing topology, snapping and splitting as needed.

Functions enhanced in PostGIS 3.5

- **ST_Clip** - Enhanced: 3.5.0 - touched argument added. Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.

Functions changed in PostGIS 3.5

- **ST_AsGeoJSON** - Changed: 3.5.0 allow specifying the column containing the feature id Return a geometry or feature in GeoJSON format.
- **ST_DFullyWithin** - Changed: 3.5.0 : the logic behind the function now uses a test of containment within a buffer, rather than the ST_MaxDistance algorithm. Results will differ from prior versions, but should be closer to user expectations. Tests if a geometry is entirely inside a distance of another

13.12.3 PostGIS Functions new or enhanced in 3.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.4

- **PostGIS_GEOS_Compiled_Version** - Availability: 3.4.0 Returns the version number of the GEOS library against which PostGIS was built.
- **PostGIS_PROJ_Compiled_Version** - Availability: 3.5.0 Returns the version number of the PROJ library against which PostGIS was built.
- **RenameTopoGeometryColumn** - Availability: 3.4.0 Renames a topogeometry column
- **RenameTopology** - Availability: 3.4.0 Renames a topology
- **ST_ClusterIntersectingWin** - Availability: 3.4.0 Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
- **ST_ClusterWithinWin** - Availability: 3.4.0 Window function that returns a cluster id for each input geometry, clustering using separation distance.
- **ST_CoverageInvalidEdges** - Availability: 3.4.0 Window function that finds locations where polygons fail to form a valid coverage.
- **ST_CoverageSimplify** - Availability: 3.4.0 Window function that simplifies the edges of a polygonal coverage.
- **ST_CoverageUnion** - Availability: 3.4.0 - requires GEOS >= 3.8.0 Computes the union of a set of polygons forming a coverage by removing shared edges.
- **ST_InverseTransformPipeline** - Availability: 3.4.0 Return a new geometry with coordinates transformed to a different spatial reference system using the inverse of a defined coordinate transformation pipeline.
- **ST_LargestEmptyCircle** - Availability: 3.4.0. Computes the largest circle not overlapping a geometry.
- **ST_LineExtend** - Availability: 3.4.0 Returns a line extended forwards and backwards by specified distances.
- **ST_TransformPipeline** - Availability: 3.4.0 Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.
- **TopoElement** - Availability: 3.4.0 Converts a topogeometry to a topoelement.
- **debug_standardize_address** - Availability: 3.4.0 Returns a json formatted text listing the parse tokens and standardizations
- **postgis_srs** - Availability: 3.4.0 Return a metadata record for the requested authority and srid.
- **postgis_srs_all** - Availability: 3.4.0 Return metadata records for every spatial reference system in the underlying Proj database.

- **postgis_srs_codes** - Availability: 3.4.0 Return the list of SRS codes associated with the given authority.
- **postgis_srs_search** - Availability: 3.4.0 Return metadata records for projected coordinate systems that have areas of usage that fully contain the bounds parameter.

Functions enhanced in PostGIS 3.4

- **PostGIS_Full_Version** - Enhanced: 3.4.0 now includes extra PROJ configurations NETWORK_ENABLED, URL_ENDPOINT and DATABASE_PATH of proj.db location Reports full PostGIS version and build configuration infos.
- **PostGIS_PROJ_Version** - Enhanced: 3.4.0 now includes NETWORK_ENABLED, URL_ENDPOINT and DATABASE_PATH of proj.db location Returns the version number of the PROJ4 library.
- **ST_AsSVG** - Enhanced: 3.4.0 to support all curve types Returns SVG path data for a geometry.
- **ST_ClosestPoint** - Enhanced: 3.4.0 - Support for geography. Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.
- **ST_LineSubstring** - Enhanced: 3.4.0 - Support for geography was introduced. Returns the part of a line between two fractional locations.
- **ST_Project** - Enhanced: 3.4.0 Allow geometry arguments and two-point form omitting azimuth. Returns a point projected from a start point by a distance and bearing (azimuth).
- **ST_Resample** - Enhanced: 3.4.0 max and min resampling options added `ST_Resample(geometry, geometry, 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.

13.12.4 PostGIS Functions new or enhanced in 3.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.3

- **RemoveUnusedPrimitives** - Availability: 3.3.0 Removes topology primitives which not needed to define existing TopoGeometry objects.
- **ST_3DConvexHull** - Availability: 3.3.0 `ST_3DConvexHull(geometry)`.
- **ST_3DUnion** - Availability: 3.3.0 aggregate variant was added Perform 3D union.
- **ST_AsMARC21** - Availability: 3.3.0 Returns geometry as a MARC21/XML record with a geographic datafield (034).
- **ST_GeomFromMARC21** - Availability: 3.3.0, requires libxml2 2.6+ Takes MARC21/XML geographic data as input and returns a PostGIS geometry object.

- ## Functions enhanced in PostGIS 3.3

- ## Functions changed in PostGIS 3.3

- ### 13.12.5 PostGIS Functions new or enhanced in 3.2

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 `[[{"geom": "LINESTRING(0 0, 1 0)", "id": 1}, {"geom": "LINESTRING(1 0, 1 1)", "id": 2}, {"geom": "LINESTRING(1 1, 0 1)", "id": 3}, {"geom": "LINESTRING(0 1, 0 0)", "id": 4}, {"geom": "LINESTRING(0 0, 1 0)", "id": 5}]]`.
- **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 函数, 用于获取 GDAL 的 VSI 选项。

Functions enhanced in PostGIS 3.2

- **GetFaceByPoint** - Enhanced: 3.2.0 more efficient implementation and clearer contract, stops working with invalid topologies. Finds face intersecting a given point.
- **ST_ClusterKMeans** - Enhanced: 3.2.0 Support for max_radius Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST_MakeValid** - Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS >= 3.10.0. Attempts to make an invalid geometry valid without losing vertices.
- **ST_PixelAsCentroid** - 2.1.0 返回 C 语言中的像素坐标。返回一个点 (x, y)。
- **ST_PixelAsCentroids** - 2.1.0 返回 C 语言中的像素坐标。返回一个点 (x, y) 的 X, Y 坐标。
- **ST_Point** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y and SRID values.
- **ST_PointM** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, M and SRID values.
- **ST_PointZ** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z and SRID values.
- **ST_PointZM** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z, M and SRID values.
- **ST_RemovePoint** - Enhanced: 3.2.0 Remove a point from a linestring.
- **ST_RemoveRepeatedPoints** - Enhanced: 3.2.0 Returns a version of a geometry with duplicate points removed.
- **ST_StartPoint** - Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString. Returns the first point of a LineString.
- **ST_Value** - 2.1.0 返回像素值。exclude_nodata_value 选项。columnx, rowy 选项。exclude_nodata_value 选项。columnx, rowy 选项。exclude_nodata_value 选项。
- **TopoGeo_AddLineString** - Enhanced: 3.2.0 added support for returning signed identifier. Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.

Functions changed in PostGIS 3.2

- **ST_Boundary** - Changed: 3.2.0 support for TIN, does not use geos, does not linearize curves `ST_Boundary(geometry)`.
- **ValidateTopology** - Changed: 3.2.0 added optional bbox parameter, perform face labeling and edge linking checks. Returns a set of `validate_topology_return_type` objects detailing issues with topology.

13.12.6 PostGIS Functions new or enhanced in 3.1

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.1

- **ST_Hexagon** - 2.1.0 `ST_Hexagon(edge_size, cell_coord)`. Returns a single hexagon, using the provided edge size and cell coordinate within the hexagon grid space.
- **ST_HexagonGrid** - 2.1.0 `ST_HexagonGrid(geometry, edge_size, cell_coord)`. Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.
- **ST_MaximumInscribedCircle** - Availability: 3.1.0. `ST_MaximumInscribedCircle(geometry)`.
- **ST_ReducePrecision** - Availability: 3.1.0. Returns a valid geometry with points rounded to a grid tolerance.
- **ST_Square** - 2.1.0 `ST_Square(edge_size, cell_coord)`. Returns a single square, using the provided edge size and cell coordinate within the square grid space.
- **ST_SquareGrid** - 2.1.0 `ST_SquareGrid(geometry, edge_size, cell_coord)`. Returns a set of grid squares and cell indices that completely cover the bounds of the geometry argument.

Functions enhanced in PostGIS 3.1

- **ST_AsEWKT** - Enhanced: 3.1.0 support for optional precision parameter. `ST_AsEWKT(geometry, precision)` WKT(Well-Known Text) `ST_AsEWKT(geometry, precision, SRID)`.
- **ST_ClusterKMeans** - Enhanced: 3.1.0 Support for 3D geometries and weights Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST_Difference** - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the part of geometry A that does not intersect geometry B.
- **ST_Intersection** - Enhanced: 3.1.0 accept a gridSize parameter Computes a geometry representing the shared portion of geometries A and B.
- **ST_MakeValid** - Enhanced: 3.1.0, added removal of Coordinates with NaN values. Attempts to make an invalid geometry valid without losing vertices.
- **ST_Subdivide** - Enhanced: 3.1.0 accept a gridSize parameter. Computes a rectilinear subdivision of a geometry.
- **ST_SymDifference** - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the portions of geometries A and B that do not intersect.
- **ST_TileEnvelope** - 2.0.0 `ST_TileEnvelope(geometry, SRID, zoom, x, y)`. Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.
- **ST_UnaryUnion** - Enhanced: 3.1.0 accept a gridSize parameter. Computes the union of the components of a single geometry.

- **ST_Union** - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the point-set union of the input geometries.

Functions changed in PostGIS 3.1

- **ST_Count** - 2.2.0 函数 `ST_Count(rastertable, rastercolumn, ...)` 返回指定栅格列中指定值的计数。如果指定了 `exclude_nodata_value` 1，则排除 NODATA 值。默认值为 0。
- **ST_Force3D** - Changed: 3.1.0. Added support for supplying a non-zero Z value. 函数 `ST_Force3DZ` 已弃用。
- **ST_Force3DM** - Changed: 3.1.0. Added support for supplying a non-zero M value. 函数 `ST_Force3DMZ` 已弃用。
- **ST_Force3DZ** - Changed: 3.1.0. Added support for supplying a non-zero Z value. 函数 `ST_Force3DZ` 已弃用。
- **ST_Force4D** - Changed: 3.1.0. Added support for supplying non-zero Z and M values. 函数 `ST_Force4D` 已弃用。
- **ST_Histogram** - Changed: 3.1.0 Removed `ST_Histogram(table_name, column_name)` variant. 使用 `(bin; ...)` 变体。
- **ST_Quantile** - Changed: 3.1.0 Removed `ST_Quantile(table_name, column_name)` variant. 使用 `(population) ... (quantile) ...` 变体。
- **ST_SummaryStats** - 2.2.0 函数 `ST_SummaryStats(rastertable, rastercolumn, ...)` 返回栅格列的统计摘要。返回 summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed if no band is specified.

13.12.7 PostGIS Functions new or enhanced in 3.0

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.0

- **CG_ConstrainedDelaunayTriangles** - 2.1.0 函数 `CG_ConstrainedDelaunayTriangles(geometry)`。Return a constrained Delaunay triangulation around the given input geometry.
- **ST_3DLineInterpolatePoint** - 2.1.0 函数 `ST_3DLineInterpolatePoint(geometry, float)`。Returns a point interpolated along a 3D line at a fractional location.
- **ST_ConstrainedDelaunayTriangles** - 2.1.0 函数 `ST_ConstrainedDelaunayTriangles(geometry)`。Return a constrained Delaunay triangulation around the given input geometry.
- **ST_TileEnvelope** - 2.1.0 函数 `ST_TileEnvelope(float, float, float, float)`。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 更新内容 PostGIS 更新内容.
- **ST_LocateBetween** - Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. Returns the portions of a geometry that match a measure range.
- **ST_LocateBetweenElevations** - Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. Returns the portions of a geometry that lie in an elevation (Z) range.
- **ST_Overlaps** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other
- **ST_Relate** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix
- **ST_Segmentize** - Enhanced: 3.0.0 Segmentize geometry now produces equal-length subsegments Returns a modified geometry/geography having no segment longer than a given distance.
- **ST_Touches** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have at least one point in common, but their interiors do not intersect
- **ST_Within** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of A lies in B, and their interiors have a point in common

Functions changed in PostGIS 3.0

- **PostGIS_Extensions_Upgrade** - Changed: 3.0.0 to repackage loose extensions and support postgis_raster. Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.
- **ST_3DDistance** - Changed: 3.0.0 - SFCGAL version removed 更新内容, 更新内容 (SRS 更新内容) 3 更新内容.
- **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. 更新内容.

- **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 国国国 GML 2 国国 GML 3 国国国国国国国国国国.
- **ST_Distance** - Changed: 3.0.0 - does not depend on SFCGAL anymore. 国国国国国国 3 国国国国 (longest) 国国国国国国国国国国.
- **ST_Intersection** - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the shared portion of geometries A and B.
- **ST_Intersects** - Changed: 3.0.0 SFCGAL version removed and native support for 2D TINS added. Tests if two geometries intersect (they have at least one point in common)
- **ST_Union** - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the point-set union of the input geometries.

13.12.8 PostGIS Functions new or enhanced in 2.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.5

- **PostGIS_Extensions_Upgrade** - Availability: 2.5.0 Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.
- **ST_Angle** - Availability: 2.5.0 国国国国国国 3 国国国国 (longest) 国国国国国国国国国国.
- **ST_AsHexWKB** - Availability: 2.5.0 Return the Well-Known Binary (WKB) in Hex representation of the raster.
- **ST_BandFileSize** - Availability: 2.5.0 Returns the file size of a band stored in file system. If no bandnum specified, 1 is assumed.
- **ST_BandFileTimestamp** - Availability: 2.5.0 Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.
- **ST_ChaikinSmoothing** - Availability: 2.5.0 Returns a smoothed version of a geometry, using the Chaikin algorithm
- **ST_FilterByM** - Availability: 2.5.0 Removes vertices based on their M value
- **ST_Grayscale** - Availability: 2.5.0 Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue
- **ST_LineInterpolatePoints** - Availability: 2.5.0 Returns points interpolated along a line at a fractional interval.
- **ST_OrientedEnvelope** - Availability: 2.5.0. Returns a minimum-area rectangle containing a geometry.
- **ST_QuantizeCoordinates** - Availability: 2.5.0 Sets least significant bits of coordinates to zero
- **ST_RastFromHexWKB** - Availability: 2.5.0 Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.
- **ST_RastFromWKB** - Availability: 2.5.0 Return a raster value from a Well-Known Binary (WKB) raster.
- **ST_SetBandIndex** - Availability: 2.5.0 Update the external band number of an out-db band

- **ST_RedBandPath** - 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. `%f/%f%f%f` WKT(Well-Known Text) `%f%f` SRID `%f%f%f%f%f%f%f%f`.
- **ST_BandMetaData** - Enhanced: 2.5.0 to include outdbbandnum, filesize and filetimestamp for outdb rasters. `%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f`. `%f%f%f%f%f%f%f%f%f%f` 1 `%f%f%f%f%f` `%f`.
- **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 `%f` `%f%f%f%f` PostGIS `%f%f%f%f%f%f%f`.
- **ST_GeometricMedian** - Enhanced: 2.5.0 Added support for M as weight of points. `%f%f%f%f%f` `%f%f%f` (median) `%f%f%f`.
- **ST_Intersects** - Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION. Tests if two geometries intersect (they have at least one point in common)
- **ST_OffsetCurve** - Enhanced: 2.5 - added support for GEOMETRYCOLLECTION and MULTILINESTRING Returns an offset line at a given distance and side from an input line.
- **ST_Scale** - Enhanced: 2.5.0 support for scaling relative to a local origin (origin parameter) was introduced. Scales a geometry by given factors.
- **ST_Split** - Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced. Returns a collection of geometries created by splitting a geometry by another geometry.
- **ST_Subdivide** - Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5. Computes a rectilinear subdivision of a geometry.

Functions changed in PostGIS 2.5

- **ST_GDALDrivers** - Changed: 2.5.0 - add can_read and can_write columns. Returns a list of raster formats supported by PostGIS through GDAL. Only those formats with can_write=True can be used by ST_AsGDALRaster

13.12.9 PostGIS Functions new or enhanced in 2.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.4

- **ST_AsGeobuf** - 2.2.0         . Return a Geobuf representation of a set of rows.
- **ST_AsMVT** - 2.2.0         . Aggregate function returning a MVT representation of a set of rows.
- **ST_AsMVTGeom** - 2.2.0         . Transforms a geometry into the coordinate space of a MVT tile.

- **ST_Centroid** - Availability: 2.4.0 support for geography was introduced. 返回几何图形的质心。
- **ST_ForcePolygonCCW** - 2.2.0 新特性。Orient all exterior rings counter-clockwise and all interior rings clockwise.
- **ST_ForcePolygonCW** - 2.2.0 新特性。Orient all exterior rings clockwise and all interior rings counter-clockwise.
- **ST_FrechetDistance** - Availability: 2.4.0 - requires GEOS >= 3.7.0 3 版本 (shortest) 返回最短距离。
- **ST_IsPolygonCCW** - 2.2.0 新特性。Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.
- **ST_IsPolygonCW** - 2.2.0 新特性。Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.
- **ST_MakeEmptyCoverage** - 2.2.0 新特性。Cover georeferenced area with a grid of empty raster tiles.

Functions enhanced in PostGIS 2.4

- **Loader_Generate_Nation_Script** - Enhanced: 2.4.1 zip code 5 tabulation area (zcta5) load step was fixed and when enabled, zcta5 data is loaded as a single table called zcta5_all as part of the nation script load. 新特性, 新特性。
- **Normalize_Address** - Enhanced: 2.4.0 norm_addy object includes additional fields zip4 and address_alphanumeric. 新特性, 新特性, 新特性, 新特性。norm_addy 新特性。tiger_geocoder 新特性 (TIGER 新特性) 新特性。
- **Page_Normalize_Address** - Enhanced: 2.4.0 norm_addy object includes additional fields zip4 and address_alphanumeric. 新特性, 新特性, 新特性, 新特性。norm_addy 新特性。tiger_geocoder 新特性 (TIGER 新特性) 新特性。address_standardizer 新特性。
- **Reverse_Geocode** - Enhanced: 2.4.1 if optional zcta5 dataset is loaded, the reverse_geocode function can resolve to state and zip even if the specific state data is not loaded. Refer to for details on loading zcta5 data. 新特性。include_strnum_range = true 新特性, 新特性。
- **ST_AsTWKB** - Enhanced: 2.4.0 memory and speed improvements. 新特性 TWKB(Tiny Well-Known Binary) 新特性。
- **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. 新特性。

Functions changed in PostGIS 2.4

- **=** - Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use `ST_Equals`. Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **ST_Node** - Changed: 2.4.0 this function uses `GEOSNode` internally instead of `GEOSUnaryUnion`. This may cause the resulting linestrings to have a different order and direction compared to PostGIS < 2.4. Nodes a collection of lines.

13.12.10 PostGIS Functions new or enhanced in 2.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.3

- **&&&(geometry,gidx)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- **&&&(gidx,geometry)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- **&&&(gidx,gidx)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.
- **&&(box2df,box2df)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- **&&(box2df,geometry)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- **&&(geometry,box2df)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- **@(box2df,box2df)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- **@(box2df,geometry)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- **@(geometry,box2df)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- **Populate_Topology_Layer** - 2.3.0 新特性. Adds missing entries to topology.layer table by reading metadata from topo tables.
- **ST_ClusterDBSCAN** - 2.3.0 新特性. Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- **ST_ClusterKMeans** - 2.3.0 新特性. Window function that returns a cluster id for each input geometry using the K-means algorithm.

- **ST_GeneratePoints** - 2.3.0 新特性。Generates a multipoint of random points contained in a Polygon or MultiPolygon.
- **ST_GeometricMedian** - 2.3.0 新特性。Returns the (median) point.
- **ST_MakeLine** - 2.0.0 新特性。Returns a line from a set of points.
- **ST_MinimumBoundingRadius** - 2.3.0 新特性。Returns the center point and radius of the smallest circle that contains a geometry.
- **ST_MinimumClearance** - 2.3.0 新特性。Returns the (robustness) clearance.
- **ST_MinimumClearanceLine** - 2.3.0 新特性。GEOS 3.6.0 新特性。Returns 2 minimum clearance lines.
- **ST_Normalize** - 2.3.0 新特性。Returns a normalized geometry.
- **ST_Points** - 2.3.0 新特性。Returns a set of points from a geometry.
- **ST_VoronoiLines** - 2.3.0 新特性。Returns the boundaries of the Voronoi diagram of the vertices of a geometry.
- **ST_VoronoiPolygons** - 2.3.0 新特性。Returns the cells of the Voronoi diagram of the vertices of a geometry.
- **ST_WrapX** - Availability: 2.3.0 requires GEOS X 新特性。
- **TopoGeom_addElement** - 2.3 新特性。Adds an element to the definition of a TopoGeometry.
- **TopoGeom_remElement** - 2.3 新特性。Removes an element from the definition of a TopoGeometry.
- **~(box2df,box2df)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- **~(geometry,box2df)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).

Functions enhanced in PostGIS 2.3

- **ST_Contains** - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of B lies in A, and their interiors have a point in common
- **ST_Covers** - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of B lies in A
- **ST_Expand** - Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions. Returns a bounding box expanded from another bounding box or a geometry.
- **ST_Intersects** - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if two geometries intersect (they have at least one point in common)

- **ST_Segmentize** - Enhanced: 2.3.0 Segmentize geography now produces equal-length subsegments. Returns a modified geometry/geography having no segment longer than a given distance.
- **ST_Transform** - Enhanced: 2.3.0 support for direct PROJ.4 text was introduced. Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST_Within** - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of A lies in B, and their interiors have a point in common

Functions changed in PostGIS 2.3

- **ST_PointN** - 新增: 2.3.0 新增 ST_PointN(-1 geometry) 返回 ST_LineString 的 ST_CircularString 子字符串。

13.12.11 PostGIS Functions new or enhanced in 2.2

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.2

- **<->** - 2.2.0 新增。PostgreSQL 9.1 新增 KNN 运算符。Returns the n-D distance between the A and B geometries or bounding boxes
- **ST_3DDifference** - 2.2.0 新增。3D 差集。
- **ST_3DUnion** - 2.2.0 新增。Perform 3D union.
- **ST_ApproximateMedialAxis** - 2.2.0 新增。返回几何的近似 medial axis。
- **ST_AsEncodedPolyline** - 2.2.0 新增。将几何体编码为 Polyline 格式。
- **ST_AsTWKB** - 2.2.0 新增。返回 TWKB (Tiny Well-Known Binary) 格式。
- **ST_BoundingDiagonal** - 2.2.0 新增。返回几何体的 bounding diagonal。
- **ST_CPAWithin** - 2.2.0 新增。Tests if the closest point of approach of two trajectories is within the specified distance.
- **ST_ClipByBox2D** - 2.2.0 新增。Computes the portion of a geometry falling within a rectangle.
- **ST_ClosestPointOfApproach** - 2.2.0 新增。Returns a measure at the closest point of approach of two trajectories.
- **ST_ClusterIntersecting** - 2.2.0 新增。Aggregate function that clusters input geometries into connected sets.
- **ST_ClusterWithin** - 2.2.0 新增。Aggregate function that clusters geometries by separation distance.
- **ST_CountAgg** - 2.2.0 新增。返回几何体的 count。exclude_nodata_value 参数默认为 1。exclude_nodata_value 为 0 时，NODATA 值将被排除。
- **ST_CreateOverview** - 2.2.0 新增。返回几何体的 overview。
- **ST_DistanceCPA** - 2.2.0 新增。Returns the distance between the closest point of approach of two trajectories.

- **ST_ForceCurve** - 2.2.0 更新内容。强制将曲线转换为指定的曲线精度 (upcast)。
- **ST_IsPlanar** - 2.2.0 更新内容。检查 2.1.0 版本中的平面性。
- **ST_IsSolid** - 2.2.0 更新内容。检查几何体是否为有效的体。
- **ST_IsValidTrajectory** - 2.2.0 更新内容。Tests if the geometry is a valid trajectory.
- **ST_LineFromEncodedPolyline** - 2.2.0 更新内容。从编码的多线 (polyline) 生成线。
- **ST_MakeSolid** - 2.2.0 更新内容。将几何体转换为体。支持 TIN 格式。
- **ST_MapAlgebra (callback function version)** - 2.2.0 更新内容。mask 参数。返回 1 或 0。
- **ST_MemSize** - 2.2.0 更新内容。返回几何体的内存大小 (字节)。
- **ST_RemoveRepeatedPoints** - 2.2.0 更新内容。Returns a version of a geometry with duplicate points removed.
- **ST_Retile** - 2.2.0 更新内容。将几何体重新分块。
- **ST_SetEffectiveArea** - 2.2.0 更新内容。Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.
- **ST_SimplifyVW** - 2.2.0 更新内容。Returns a simplified representation of a geometry, using the Visvalingam-Whyatt algorithm.
- **ST_Subdivide** - 2.2.0 更新内容。Computes a rectilinear subdivision of a geometry.
- **ST_SummaryStatsAgg** - 2.2.0 更新内容。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 更新内容。交换几何体的坐标。
- **ST_Volume** - 2.2.0 更新内容。3D 几何体的体积。
- **parse_address** - 2.2.0 更新内容。解析地址。
- **postgis.enable_outdb_rasters** - 2.2.0 更新内容。DB 外部栅格。
- **postgis.gdal_datapath** - 2.2.0 更新内容。GDAL 的 GDAL_DATA 路径。
- **postgis.gdal_enabled_drivers** - 2.2.0 更新内容。PostGIS 支持的 GDAL 驱动程序。
- **standardize_address** - 2.2.0 更新内容。标准化地址。
- **|=** - 2.2.0 更新内容。PostgreSQL 9.5 索引支持 (index-supported)。

Functions enhanced in PostGIS 2.2

- **<->** - **2.2.0** -- PostgreSQL 9.5 **KNN**("K nearest neighbor") **PostgreSQL 9.4** **A B** **2**.
- **AsTopoJSON** - **2.2.1** **(pungal)** **TopoGeometry** **TopoJSON**.
- **ST_Area** - **2.2.0** **GeographicLib** **Proj 4.9.0**.
- **ST_AsX3D** - **2.2.0** **(x/y, /)** **ISO-IEC-19776-1.2-X3DEncodings-XML**.
- **ST_Azimuth** - **2.2.0** **GeographicLib** **Proj 4.9.0** **2**.
- **ST_Distance** - **2.2.0** **GeographicLib** **Proj 4.9.0** **3** **(longest)**.
- **ST_Scale** - **Enhanced: 2.2.0** support for scaling all dimension (factor parameter) was introduced. Scales a geometry by given factors.
- **ST_Split** - **Enhanced: 2.2.0** support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced. Returns a collection of geometries created by splitting a geometry by another geometry.
- **ST_Summary** - **2.2.0** **TIN** **(curve)**.

Functions changed in PostGIS 2.2

- **<->** - **<->**: 2.2.0 **<->** -- PostgreSQL 9.5 **<->** (Hybrid syntax) **<->** PostgreSQL 9.5 **<->**. A **<->** B **<->** 2 **<->**.
- **Get_Geocode_Setting** - **<->**: 2.2.0 **<->** geocode_settings_default **<->** geocode_settings **<->** geocode_settings **<->** tiger.geocode_settings **<->**.
- **ST_3DClosestPoint** - **<->**: 2.2.0 **<->** 2D **<->**, (**<->** Z **<->** 0 **<->**) 2D **<->**. 2D **<->** 3D **<->**, **<->** Z **<->** Z **<->** 0 **<->**. g2 **<->** g1 **<->** 3 **<->**.
- **ST_3DDistance** - **<->**: 2.2.0 **<->**, 2D **<->** 3D **<->** Z **<->** Z **<->** 0 **<->**. **<->**, **<->** (SRS **<->**) 3 **<->**.
- **ST_3DLongestLine** - **<->**: 2.2.0 **<->** 2D **<->**, (**<->** Z **<->** 0 **<->**) 2D **<->**. 2D **<->** 3D **<->**, **<->** Z **<->** Z **<->** 0 **<->**. **<->** 3 **<->** (longest) **<->**.
- **ST_3DMaxDistance** - **<->**: 2.2.0 **<->**, 2D **<->** 3D **<->** Z **<->** Z **<->** 0 **<->**. **<->**, **<->** (SRS **<->**) 3 **<->**.
- **ST_3DShortestLine** - **<->**: 2.2.0 **<->** 2D **<->**, (**<->** Z **<->** 0 **<->**) 2D **<->**. 2D **<->** 3D **<->**, **<->** Z **<->** Z **<->** 0 **<->**. **<->** 3 **<->** (shortest) **<->**.
- **ST_DistanceSphere** - **<->**: 2.2.0 **<->** ST_Distance_Sphere **<->**. **<->** PostgreSQL 1.5 **<->**.

- **ST_DistanceSpheroid** - 更新: 2.2.0 支持 ST_Distance_Spheroid 函数。该函数在 PostGIS 1.5 中引入，用于计算球面距离。
- **ST_Equals** - 更改: 2.2.0 对于无效的几何体，如果它们是二进制相等，测试如果两个几何体包含相同的点集。
- **ST_LengthSpheroid** - 更新: 2.2.0 支持 ST_Length_Spheroid 函数，ST_3DLength_Spheroid 函数。用于计算球面长度。
- **ST_MemSize** - 更改: 2.2.0 名称更改为 ST_MemSize 以遵循命名约定。ST_Geometry 函数。
- **ST_PointInsideCircle** - 更改: 2.2.0 在以前的版本中，这被称为 ST_Point_Inside_Circle。测试如果点几何体在由中心和半径定义的圆内。
- **ValidateTopology** - 更新: 2.2.0 支持 'edge crosses node' 类型的拓扑问题。返回 id1 和 id2 的 validate_topology_return_type 对象，详细描述拓扑问题。

13.12.12 PostGIS Functions new or enhanced in 2.1

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.1

- **=** - 2.1.0 支持 A 等于 B 的 TRUE 测试。
- **AsTopoJSON** - 2.1.0 支持 TopoGeometry 到 TopoJSON 的转换。
- **Drop_Nation_Tables_Generate_Script** - 2.1.0 支持 county_all, state_all 等表，以及 county, state (州) 表。
- **Get_Geocode_Setting** - 2.1.0 支持 tiger.geocode_settings 表。
- **Loader_Generate_Nation_Script** - 2.1.0 支持 county, state 表。
- **Page_Normalize_Address** - 2.1.0 支持 norm_addy 函数，tiger_geocoder (TIGER 索引) 和 address_standardizer 函数。
- **ST_3DArea** - 2.1.0 支持 3 维面积计算。返回 0 或 NaN。
- **ST_3DIntersection** - 2.1.0 支持 3 维交集计算。
- **ST_Box2dFromGeoHash** - 2.1.0 支持 GeoHash 到 BOX2D 的转换。
- **ST_ColorMap** - 2.1.0 支持 8BUI 表 (grayscale, RGB, RGBA) 和 4 个颜色通道。返回 1 个颜色值。
- **ST_Contains** - 2.1.0 支持 rastA 和 rastB 的包含测试。
- **ST_ContainsProperly** - 2.1.0 支持 rastB 包含 rastA 的测试。

- **ST_CoveredBy** - 2.1.0 函数。返回栅格 rastA 是否被栅格 rastB 完全覆盖。
- **ST_Covers** - 2.1.0 函数。返回栅格 rastB 是否完全覆盖栅格 rastA。
- **ST_DFullyWithin** - 2.1.0 函数。返回栅格 rastA 是否完全在栅格 rastB 的范围内。
- **ST_DWithin** - 2.1.0 函数。返回栅格 rastA 是否与栅格 rastB 在指定距离内相交。
- **ST_DelaunayTriangles** - 2.1.0 函数。Returns the Delaunay triangulation of the vertices of a geometry.
- **ST_Disjoint** - 2.1.0 函数。返回栅格 rastA 是否与栅格 rastB 不相交。
- **ST_DumpValues** - 2.1.0 函数。返回栅格的所有值。
- **ST_Extrude** - 2.1.0 函数。将栅格沿 Z 轴拉伸。
- **ST_ForceLHR** - 2.1.0 函数。LHR(Left Hand Reverse; 左手规则) 强制栅格。
- **ST_FromGDALRaster** - 2.1.0 函数。从 GDAL 栅格创建 PostGIS 栅格。
- **ST_GeomFromGeoHash** - 2.1.0 函数。GeoHash 字符串转换为几何。
- **ST_InvDistWeight4ma** - 2.1.0 函数。4 邻域反距离权重。
- **ST_MapAlgebra (callback function version)** - 2.1.0 函数。使用回调函数进行栅格映射代数。
- **ST_MapAlgebra (expression version)** - 2.1.0 函数。使用 SQL 表达式进行栅格映射代数。
- **ST_MinConvexHull** - 2.1.0 函数。返回栅格的最小凸包。
- **ST_MinDist4ma** - 2.1.0 函数。4 邻域最小距离。
- **ST_MinkowskiSum** - 2.1.0 函数。闵可夫斯基和。
- **ST_NearestValue** - 2.1.0 函数。返回栅格中最近的值。
- **ST_Neighborhood** - 2.1.0 函数。返回栅格的邻域。
- **ST_NotSameAlignmentReason** - 2.1.0 函数。返回栅格不同对齐的原因。
- **ST_Orientation** - 2.1.0 函数。返回栅格的朝向 (orientation)。
- **ST_Overlaps** - 2.1.0 函数。返回栅格 rastA 是否与栅格 rastB 重叠。
- **ST_PixelAsCentroid** - 2.1.0 函数。返回栅格的像素作为质心 (centroid)。

- **ST_PixelAsCentroids** - 2.1.0 返回栅格像素的质心。返回的几何体是点 (POINT) 类型。返回的几何体 X, Y 坐标是像素的质心。返回的几何体是点 (POINT) 类型。
- **ST_PixelAsPoint** - 2.1.0 返回栅格像素的点。返回的几何体是点 (POINT) 类型。
- **ST_PixelAsPoints** - 2.1.0 返回栅格像素的点。返回的几何体是点 (POINT) 类型。返回的几何体 X, Y 坐标是像素的质心。返回的几何体是点 (POINT) 类型。
- **ST_PixelOfValue** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体 columnx, rowy 是像素的坐标。
- **ST_PointFromGeoHash** - 2.1.0 返回 GeoHash 点。返回的几何体是点 (POINT) 类型。
- **ST_RasterToWorldCoord** - 2.1.0 返回栅格像素的世界坐标。返回的几何体是点 (POINT) 类型。返回的几何体 X, Y(经度, 纬度) 是像素的坐标。返回的几何体是点 (POINT) 类型。
- **ST_Resize** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体 GDAL 1.6.1 返回的几何体是点 (POINT) 类型。
- **ST_Roughness** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体 DEM 返回的几何体 (roughness) 是栅格像素的粗糙度。
- **ST_SetValues** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体是点 (POINT) 类型。
- **ST_Simplify** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体 TopoGeometry 返回的几何体是点 (POINT) 类型。
- **ST_StraightSkeleton** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体 (straight skeleton) 是栅格像素的骨架。
- **ST_Summary** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体是点 (POINT) 类型。
- **ST_TPI** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体 (Topographic Position Index) 是栅格像素的拓扑位置指数。
- **ST_TRI** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体 (Terrain Ruggedness Index) 是栅格像素的地形粗糙度指数。
- **ST_Tessellate** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体 (tessellation) 是栅格像素的三角网。
- **ST_Tile** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体是点 (POINT) 类型。
- **ST_Touches** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体 rastA 返回的几何体 rastB 返回的几何体是点 (POINT) 类型。
- **ST_Union** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体 ST_Union(rast, unionarg) 返回的几何体是点 (POINT) 类型。
- **ST_Within** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体 rastB 返回的几何体 rastA 返回的几何体是点 (POINT) 类型。
- **ST_WorldToRasterCoord** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体 X, Y(经度, 纬度) 返回的几何体是点 (POINT) 类型。
- **Set_Geocode_Setting** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体是点 (POINT) 类型。
- **UpdateRasterSRID** - 2.1.0 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体 SRID 返回的几何体是点 (POINT) 类型。
- **clearTopoGeom** - 2.1 返回栅格像素。返回的几何体是点 (POINT) 类型。返回的几何体 Clears the content of a topo geometry.

- **postgis_sfcgal_version** - 2.1.0

Functions enhanced in PostGIS 2.1

- **ST_AddBand** - `addband`: 2.1.0 `addbandarg` added. `addbandarg` can now be used to specify the band number to add. If not specified, the first band is added.
- **ST_AddBand** - `addband`: 2.1.0 `addbandarg` DB support added. `addbandarg` can now be used to specify the band number to add. If not specified, the first band is added.
- **ST_AsBinary/ST_AsWKB** - `outasbin`: 2.1.0 `outasbin` added. Return the Well-Known Binary (WKB) representation of the raster.
- **ST_AsGML** - `gmlid`: 2.1.0 `gmlid` GML 3 ID support added. `gmlid` GML 2 or GML 3 support added.
- **ST_Aspect** - `interpolate_nodata`: 2.1.0 `interpolate_nodata` added. `interpolate_nodata` can be used to specify whether to interpolate nodata values. If not specified, the default behavior is to not interpolate.
- **ST_Boundary** - `boundary`: 2.1.0 `boundary` added. `boundary` can be used to specify the boundary type. If not specified, the default behavior is to return the outer boundary.
- **ST_Clip** - `cclip`: 2.1.0 `cclip` C support 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.
- **ST_DWithin** - Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details. Tests if two geometries are within a given distance
- **ST_DWithin** - Enhanced: 2.1.0 support for curved geometries was introduced. Tests if two geometries are within a given distance
- **ST_Distance** - Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details. `distance` 3 (longest) support added.
- **ST_Distance** - `distance`: 2.1.0 `distance` 3 (longest) support added.
- **ST_Distinct4ma** - `distinct4ma`: 2.1.0 `distinct4ma` 2 support added. `distinct4ma` can be used to specify the distinctness parameter. If not specified, the default behavior is to return the distinctness parameter.
- **ST_DumpPoints** - Enhanced: 2.1.0 Faster speed. Reimplemented as native-C. `dumppoints` support added.
- **ST_HillShade** - `interpolate_nodata`: 2.1.0 `interpolate_nodata` added. `interpolate_nodata` can be used to specify whether to interpolate nodata values. If not specified, the default behavior is to not interpolate.
- **ST_MakeValid** - Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT. Attempts to make an invalid geometry valid without losing vertices.
- **ST_Max4ma** - `max4ma`: 2.1.0 `max4ma` 2 support added. `max4ma` can be used to specify the maximum value. If not specified, the default behavior is to return the maximum value.
- **ST_Mean4ma** - `mean4ma`: 2.1.0 `mean4ma` 2 support added. `mean4ma` can be used to specify the mean value. If not specified, the default behavior is to return the mean value.
- **ST_Min4ma** - `min4ma`: 2.1.0 `min4ma` 2 support added. `min4ma` can be used to specify the minimum value. If not specified, the default behavior is to return the minimum value.

- ST_PixelAsPolygons** - **2.1.0** `ST_PixelAsPolygons(raster, exclude nodata value)`. Returns a set of polygons representing the pixels of the raster. The polygons are returned in the order of the pixels in the raster. The polygons are returned in the order of the pixels in the raster. The polygons are returned in the order of the pixels in the raster.
- ST_Polygon** - **2.1.0** `ST_Polygon(geomval)`. Returns a polygon geometry from a geometry value. The geometry value can be a geometry object or a geometry value. The geometry value can be a geometry object or a geometry value. The geometry value can be a geometry object or a geometry value.
- ST_Range4ma** - **2.1.0** `ST_Range4ma(raster, distance)`. Returns a raster with the range of values in the input raster for a given distance. The distance is in units of the raster's resolution. The distance is in units of the raster's resolution. The distance is in units of the raster's resolution.
- ST_SameAlignment** - **2.1.0** `ST_SameAlignment(raster1, raster2)`. Returns a boolean value indicating if two rasters have the same alignment. The rasters must have the same resolution and the same origin. The rasters must have the same resolution and the same origin. The rasters must have the same resolution and the same origin.
- ST_Segmentize** - **2.1.0** `ST_Segmentize(geometry, distance)`. Returns a modified geometry/geography having no segment longer than a given distance. The distance is in units of the geometry's SRID. The distance is in units of the geometry's SRID. The distance is in units of the geometry's SRID.
- ST_SetGeoReference** - **2.1.0** `ST_SetGeoReference(raster, double precision, ...)`. Sets the geographic reference information for a raster. The geographic reference information is stored in the raster's metadata. The geographic reference information is stored in the raster's metadata. The geographic reference information is stored in the raster's metadata.
- ST_SetValue** - **2.1.0** `ST_SetValue(raster, geomval)`. Sets the value of a pixel in a raster to a given geometry value. The geometry value can be a geometry object or a geometry value. The geometry value can be a geometry object or a geometry value. The geometry value can be a geometry object or a geometry value.
- ST_Slope** - **2.1.0** `ST_MapAlgebra(raster, units, scale, interpolate_nodata)`. Returns a raster with the slope of the input raster. The slope is in units of the raster's resolution. The slope is in units of the raster's resolution. The slope is in units of the raster's resolution.
- ST_StdDev4ma** - **2.1.0** `ST_StdDev4ma(raster, distance)`. Returns a raster with the standard deviation of the input raster for a given distance. The distance is in units of the raster's resolution. The distance is in units of the raster's resolution. The distance is in units of the raster's resolution.
- ST_Sum4ma** - **2.1.0** `ST_Sum4ma(raster, distance)`. Returns a raster with the sum of the input raster for a given distance. The distance is in units of the raster's resolution. The distance is in units of the raster's resolution. The distance is in units of the raster's resolution.
- ST_Summary** - **2.1.0** `ST_Summary(raster)`. Returns a summary of the input raster. The summary includes the number of pixels, the minimum and maximum values, and the standard deviation. The summary includes the number of pixels, the minimum and maximum values, and the standard deviation. The summary includes the number of pixels, the minimum and maximum values, and the standard deviation.
- ST_Transform** - **2.1.0** `ST_Transform(raster, alignto)`. Returns a raster with the input raster transformed to a given alignment. The alignment is in units of the raster's resolution. The alignment is in units of the raster's resolution. The alignment is in units of the raster's resolution.
- ST_Union** - **2.1.0** `ST_Union(raster)`. Returns a union of the input raster. The union is a single geometry object. The union is a single geometry object. The union is a single geometry object.
- ST_Union** - **2.1.0** `ST_Union(raster, uniontype)`. Returns a union of the input raster. The union is a single geometry object. The union is a single geometry object. The union is a single geometry object.
- toTopoGeom** - **2.1.0** `toTopoGeom(geometry)`. Converts a simple Geometry into a topo geometry. The topo geometry is a geometry object. The topo geometry is a geometry object. The topo geometry is a geometry object.

Functions changed in PostGIS 2.1

- **ST_Aspect** - : 2.1.0                             . 2.1.0 . . .

- 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** - `geometry`: 2.1.0 `geometry`, `2.0.x geometry` ST_Force_2D `geometry`. `2D geometry` `geometry`.
- ST_Force3D** - `geometry`: 2.1.0 `geometry`, `2.0.x geometry` ST_Force_3D `geometry`. `XYZ geometry`. ST_Force3DZ `geometry`.
- ST_Force3DM** - `geometry`: 2.1.0 `geometry`, `2.0.x geometry` ST_Force_3DM `geometry`. `XYM geometry`.
- ST_Force3DZ** - `geometry`: 2.1.0 `geometry`, `2.0.x geometry` ST_Force_3DZ `geometry`. `XYZ geometry`.
- ST_Force4D** - `geometry`: 2.1.0 `geometry`, `2.0.x geometry` ST_Force_4D `geometry`. `XYZM geometry`.
- ST_ForceCollection** - `geometry`: 2.1.0 `geometry`, `2.0.x geometry` ST_Force_Collection `geometry`. `geometry`.
- ST_HillShade** - `geometry`: 2.1.0 `geometry`. 2.1.0 `geometry` `geometry`. `geometry`, `geometry`, `geometry` `geometry`.
- ST_LineInterpolatePoint** - `geometry`: 2.1.0 `geometry`, `2.0.x geometry` ST_Line_Interpolate_Point `geometry`. Returns a point interpolated along a line at a fractional location.
- ST_LineLocatePoint** - `geometry`: 2.1.0 `geometry`, `2.0.x geometry` ST_Line_Locate_Point `geometry`. Returns the fractional location of the closest point on a line to a point.
- ST_LineSubstring** - `geometry`: 2.1.0 `geometry`, `2.0.x geometry` ST_Line_Substring `geometry`. Returns the part of a line between two fractional locations.
- ST_PixelAsCentroids** - `geometry`: 2.1.1 `geometry` exclude_nodata_value `geometry`. `geometry` (`geometry`) `geometry` X, Y `geometry`. `geometry`.
- ST_PixelAsPoints** - `geometry`: 2.1.1 `geometry` exclude_nodata_value `geometry`. `geometry` `geometry` X, Y `geometry`. `geometry`.
- ST_PixelAsPolygons** - `geometry`: 2.1.1 `geometry` exclude_nodata_value `geometry`. `geometry` `geometry` X, Y `geometry`.
- ST_Polygon** - `geometry`: 2.1.0 `geometry`, `geometry` `geometry`. NODATA `geometry`.
- ST_RasterToWorldCoordX** - `geometry`: 2.1.0 `geometry` ST_Raster2WorldCoordX `geometry`. `geometry` X `geometry`. `geometry` 1 `geometry`.
- ST_RasterToWorldCoordY** - `geometry`: 2.1.0 `geometry` ST_Raster2WorldCoordY `geometry`. `geometry` Y `geometry`. `geometry` 1 `geometry`.
- ST_Rescale** - `geometry`: 2.1.0 `geometry` SRID `geometry`. 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** - `geometry`: 2.1.0 `geometry` SRID `geometry`. `geometry` (`geometry`) `geometry`. NearestNeighbor(`geometry`), Bilinear, Cubic, CubicSpline `geometry` `geometry` Lanczos `geometry`. `geometry` NearestNeighbor `geometry`.

- **ST_Segmentize** - Changed: 2.1.0 As a result of the introduction of geography support, the usage `ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5)` causes an ambiguous function error. The input needs to be properly typed as a geometry or geography. Use `ST_GeomFromText`, `ST_GeogFromText` or a cast to the required type (e.g. `ST_Segmentize('LINESTRING(1 2, 3 4)::geometry, 0.5)`) Returns a modified geometry/geography having no segment longer than a given distance.
- **ST_Slope** - 新增: 2.1.0 返回给定几何体的坡度。2.1.0 返回给定几何体的坡度。返回类型: `float`。返回类型: `float`。
- **ST_SnapToGrid** - 新增: 2.1.0 将几何体 snap 到指定的 SRID 网格上。返回类型: `geometry`。最近邻 (NearestNeighbor), Bilinear, Cubic, CubicSpline 和 Lanczos 插值方法。最近邻 (NearestNeighbor) 插值方法。
- **ST_WorldToRasterCoordX** - 新增: 2.1.0 将世界坐标转换为栅格坐标 X。返回类型: `float` (pt) 返回类型: `float` (pt) X, Y 坐标 (xw, yw) 返回类型: `float`。
- **ST_WorldToRasterCoordY** - 新增: 2.1.0 将世界坐标转换为栅格坐标 Y。返回类型: `float` (pt) 返回类型: `float` (pt) X, Y 坐标 (xw, yw) 返回类型: `float`。

13.12.13 PostGIS Functions new or enhanced in 2.0

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.0

- **&&** - 2.0.0 返回两个几何体是否相交。A 和 B 相交返回 TRUE。
- **&&&** - 2.0.0 返回两个几何体是否相交。A 和 B 相交返回 TRUE。
- **<#>** - 2.0.0 返回两个几何体是否相交。PostgreSQL 9.1 返回 KNN 索引。A 和 B 相交返回 2 个索引。
- **<->** - 2.0.0 返回两个几何体之间的最短距离。返回 KNN 索引。PostgreSQL 9.1 返回 KNN 索引。A 和 B 相交返回 2 个索引。
- **@** - 2.0.0 返回 raster @ raster, raster @ geometry 是否相交。B 和 A 相交返回 TRUE。
- **@** - 2.0.5 返回 geometry @ raster 是否相交。B 和 A 相交返回 TRUE。
- **AddEdge** - 2.0.0 添加新的边。返回 ID(edgeid)。
- **AddFace** - 2.0.0 添加新的面。返回 ID(faceid)。
- **AddNode** - 2.0.0 添加新的节点。返回 ID(nodeid)。
- **AddOverviewConstraints** - 2.0.0 添加概述约束。返回 ID(overview)。

- AddRasterConstraints** - 2.0.0 `add_raster_constraints(table, spatial_ref, scaling, blocksize, alignment, bands, band_type, flag)`. 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 `as_gml(geometry)`. TopoGeometry to GML geometry.
- CopyTopology** - 2.0.0 `copy_topology(topology)`. Makes a copy of a topology (nodes, edges, faces, layers and TopoGeometries) into a new schema.
- DropOverviewConstraints** - 2.0.0 `drop_overview_constraints(overview)`. Drops overview constraints.
- DropRasterConstraints** - 2.0.0 `drop_raster_constraints(raster_name)`. Drops PostGIS raster constraints from a raster table.
- DropIndexesGenerateScript** - 2.0.0 `drop_indexes_generate_script(tiger_data)`. TIGER indexes generate script for tiger_data.
- DropStateTablesGenerateScript** - 2.0.0 `drop_state_tables_generate_script(tiger_data)`. State tables generate script for tiger_data.
- GeocodeIntersection** - 2.0.0 `geocode_intersection(geomout, normalized_address)`. NAD83 geocode intersection geomout, normalized address (addy). Returns a set of topoelement objects containing the topological element_id,element_type of the given TopoGeometry (primitive elements).
- GetEdgeByPoint** - 2.0.0 `get_edge_by_point(point)`. Finds the edge-id of an edge that intersects a given point.
- GetFaceByPoint** - 2.0.0 `get_face_by_point(point)`. Finds face intersecting a given point.
- GetNodeByPoint** - 2.0.0 `get_node_by_point(point)`. Finds the node-id of a node at a point location.
- GetNodeEdges** - 2.0 `get_node_edges(node_id)`.
- GetRingEdges** - 2.0.0 `get_ring_edges(face_id)`.
- GetTopoGeomElements** - 2.0.0 `get_topo_geom_elements(topogeom)`. Returns a set of topoelement objects containing the topological element_id,element_type of the given TopoGeometry (primitive elements).
- GetTopologySRID** - 2.0.0 `get_topology_srid(topology.topology)`. Returns SRID.
- GetTract** - 2.0.0 `get_tract(tract, field)`. Returns tract (tract) field (field).
- InstallMissingIndexes** - 2.0.0 `install_missing_indexes(join, key)`.
- LoaderGenerateCensusScript** - 2.0.0 `loader_generate_census_script(state, tiger_data)`. Loader generate census script (state), TIGER (state) (tract), bg, tabblock tiger_data.

- **Loader_Generate_Script** - 2.0.0 [加载生成脚本](#). TIGER 2010 [数据源](#) (`tract`), [块面](#) (`bg`), [表块面](#) (`tabblock`) [数据集](#). [地理名称](#) (`州`) [列表](#), TIGER [数据集](#) `tiger_data` [属性表](#). [州](#) (`州`) [列表](#) [属性表](#). [TIGER 2010](#) [数据源](#), [块面](#), [表块面](#).
- **Missing_Indexes_Generate_Script** - 2.0.0 [缺失索引生成脚本](#). [连接](#) (`join`) [键](#) (`key`) [SQL DDL](#) [语句](#).
- **Polygonize** - 2.0.0 [多边形化](#). Finds and registers all faces defined by topology edges.
- **Reverse_Geocode** - 2.0.0 [反向地址编码](#). [包含字符串范围](#) = true [选项](#), [包含字符串范围](#).
- **ST_3DClosestPoint** - 2.0.0 [最近点](#). g2 [距离](#) g1 [距离](#) 3 [维空间](#) [内积](#). [3D](#) [距离](#).
- **ST_3DDFullyWithin** - 2.0.0 [完全在内部](#). Tests if two 3D geometries are entirely within a given 3D distance
- **ST_3DDWithin** - 2.0.0 [在内部](#). Tests if two 3D geometries are within a given 3D distance
- **ST_3DDistance** - 2.0.0 [距离](#). [SRS](#) [坐标系统](#) 3 [维空间](#) [距离](#).
- **ST_3DIntersects** - 2.0.0 [相交](#). Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- **ST_3DLongestLine** - 2.0.0 [最长线](#). [3](#) [维空间](#) (longest) [线段](#).
- **ST_3DMaxDistance** - 2.0.0 [最大距离](#). [SRS](#) [坐标系统](#) 3 [维空间](#) [最大距离](#).
- **ST_3DShortestLine** - 2.0.0 [最短线](#). [3](#) [维空间](#) (shortest) [线段](#).
- **ST_AddEdgeModFace** - 2.0 [添加边修改面](#). [添加边](#), [修改面](#).
- **ST_AddEdgeNewFaces** - 2.0 [添加边新建面](#). [添加边](#), [新建面](#) 2 [维空间](#).
- **ST_AsGDALRaster** - 2.0.0 [转换为GDAL栅格](#). GDAL 1.6.0 [格式](#). 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 [转换为JPEG](#). GDAL 1.6.0 [格式](#). [联合摄影测量专家组](#) JPEG(Joint Photographic Exports Group) [格式](#) ([联合摄影测量专家组](#)) [格式](#). [1](#) [维空间](#) 3 [维空间](#) RGB [颜色模型](#).
- **ST_AsLatLonText** - 2.0 [转换为经纬度文本](#). [经纬度](#), [逗号](#), [空格](#).
- **ST_AsPNG** - 2.0.0 [转换为PNG](#). GDAL 1.6.0 [格式](#). [便携式网络图形](#) PNG(Portable Network Graphics) [格式](#) ([便携式网络图形](#)) [格式](#). [1](#) [维空间](#) 3 [维空间](#) RGB [颜色模型](#) RGBA [颜色模型](#).
- **ST_AsRaster** - 2.0.0 [转换为栅格](#). GDAL 1.6.0 [格式](#). PostGIS [栅格](#) [函数](#).

- **ST_AsTIFF** - 2.0.0 函数。GDAL 1.6.0 函数。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 函数。ISO-IEC-19776-1.2-X3DEncodings-XML 函数。X3D XML 函数: ISO-IEC-19776-1.2-X3DEncodings-XML 函数。
- **ST_Aspect** - 2.0.0 函数。函数 (函数) 函数。函数。
- **ST_Band** - 2.0.0 函数。函数。函数。
- **ST_BandIsNoData** - 2.0.0 函数。函数 NODATA 函数。
- **ST_Clip** - 2.0.0 函数。Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.
- **ST_CollectionHomogenize** - 2.0.0 函数。Returns the simplest representation of a geometry collection.
- **ST_ConcaveHull** - 2.0.0 函数。Computes a possibly concave geometry that contains all input geometry vertices
- **ST_Count** - 2.0.0 函数。函数。函数。1 函数。exclude_nodata_value 函数, NODATA 函数。
- **ST_CreateTopoGeo** - 2.0 函数。函数。
- **ST_Distinct4ma** - 2.0.0 函数。函数。
- **ST_FlipCoordinates** - 2.0.0 函数。Returns a version of a geometry with X and Y axis flipped.
- **ST_GDALDrivers** - 2.0.0 函数。GDAL 1.6.0 函数。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 函数。JSON-C 0.9 函数。GeoJSON 函数 PostGIS 函数。
- **ST_GetFaceEdges** - 2.0 函数。aface 函数。
- **ST_HasNoBand** - 2.0.0 函数。函数。函数。1 函数。
- **ST_HillShade** - 2.0.0 函数。函数, 函数, 函数。
- **ST_Histogram** - 2.0.0 函数。函数 (bin; 函数) 函数。函数。
- **ST_InterpolatePoint** - 2.0.0 函数。函数 (M 函数) 函数。

- **ST_IsEmpty** - 2.0.0 返回一个布尔值。如果几何体 (width = 0, height = 0) 是空的，则返回 true。否则返回 false。
- **ST_IsValidDetail** - 2.0.0 返回一个 valid_detail 行，说明几何体是否有效，或无效的原因和位置。
- **ST_IsValidReason** - Availability: 2.0 version taking flags. Returns text stating if a geometry is valid, or a reason for invalidity.
- **ST_MakeLine** - 2.0.0 返回一个由两个点组成的线。如果两个点相同，则返回一个点。如果两个点不同，则返回一条线。
- **ST_MakeValid** - 2.0.0 尝试使无效的几何体有效，而不会丢失顶点。
- **ST_MapAlgebraExpr** - 2.0.0 返回一个几何体。它接受 1 个参数：PostgreSQL 表达式，返回 1 个几何体。如果表达式返回 NULL，则返回 NULL。
- **ST_MapAlgebraExpr** - 2.0.0 返回一个几何体。它接受 2 个参数：PostgreSQL 表达式，返回 1 个几何体。如果表达式返回 NULL，则返回 NULL。extenttype 可以是 INTERSECTION, UNION, FIRST, SECOND 之一。
- **ST_MapAlgebraFct** - 2.0.0 返回一个几何体。它接受 1 个参数：PostgreSQL 表达式，返回 1 个几何体。如果表达式返回 NULL，则返回 NULL。
- **ST_MapAlgebraFct** - 2.0.0 返回一个几何体。它接受 2 个参数：PostgreSQL 表达式，返回 1 个几何体。如果表达式返回 NULL，则返回 NULL。extenttype 可以是 INTERSECTION 之一。
- **ST_MapAlgebraFctNgb** - 2.0.0 返回一个几何体。它接受 1 个参数：PostgreSQL 表达式 (Map Algebra Nearest Neighbor) 返回 1 个几何体。如果表达式返回 NULL，则返回 NULL。
- **ST_Max4ma** - 2.0.0 返回一个几何体。它接受 4 个几何体，返回它们的最大值。
- **ST_Mean4ma** - 2.0.0 返回一个几何体。它接受 4 个几何体，返回它们的平均值。
- **ST_Min4ma** - 2.0.0 返回一个几何体。它接受 4 个几何体，返回它们的最小值。
- **ST_ModEdgeHeal** - 2.0 修复两条边，通过删除连接它们的节点，修改第一条边并删除第二条边。返回删除的节点的 id。
- **ST_MoveIsoNode** - 2.0.0 移动一个孤立节点。如果新的点几何体存在，则移动该节点。如果不存在，则返回错误。返回移动的描述。
- **ST_NewEdgeHeal** - 2.0 修复两条边，通过删除连接它们的节点，并用一条具有相同方向的新边替换它们。
- **ST_Node** - 2.0.0 返回一个几何体。它接受一个几何体，返回其节点集合。
- **ST_NumPatches** - 2.0.0 返回一个整数。它接受一个几何体，返回其补丁数量。如果几何体是 NULL，则返回 NULL。
- **ST_OffsetCurve** - 2.0 返回一个偏移线。它接受一个距离和一个侧向。返回一个输入线的偏移线。
- **ST_PatchN** - 2.0.0 返回一个几何体。它接受 ST_Geometry 返回一个几何体。

- **ST_Perimeter** - 2.0.0 新增。Returns the length of the boundary of a polygonal geometry or geography.
- **ST_PixelAsPolygon** - 2.0.0 新增。Returns a polygon from a pixel value in a raster.
- **ST_PixelAsPolygons** - 2.0.0 新增。Returns a set of polygons from a raster.
- **ST_Project** - 2.0.0 新增。Returns a point projected from a start point by a distance and bearing (azimuth).
- **ST_Quantile** - 2.0.0 新增。Returns the quantile (population) of a raster.
- **ST_Range4ma** - 2.0.0 新增。Returns a 4-metric range from a raster.
- **ST_Reclass** - 2.0.0 新增。Returns a raster with the same bands as the input, but with the values reclassified according to the nband.
- **ST_RelateMatch** - 2.0.0 新增。Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern.
- **ST_RemEdgeModFace** - 2.0 新增。Removes an edge, and if the edge separates two faces deletes one face and modifies the other face to cover the space of both.
- **ST_RemEdgeNewFace** - 2.0 新增。Removes an edge, and if the edge separates two faces creates a new face and modifies the other face to cover the space of both.
- **ST_Resample** - 2.0.0 新增。GDAL 1.6.1 新增。Resample a raster by adjusting only its scale (or pixel size).
- **ST_Rescale** - 2.0.0 新增。GDAL 1.6.1 新增。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。GDAL 1.6.1 新增。Reskew a raster (NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos, NearestNeighbor).
- **ST_SameAlignment** - 2.0.0 新增。Returns true if two rasters have the same alignment.
- **ST_SetBandIsNoData** - 2.0.0 新增。Sets the isnodata flag for a band in a raster.
- **ST_SharedPaths** - 2.0.0 新增。Returns a set of shared paths between two rasters.
- **ST_Slope** - 2.0.0 新增。Returns the slope (in degrees) of a raster.
- **ST_Snap** - 2.0.0 新增。Returns a raster with the same bands as the input, but with the values snapped to the nearest integer.

- ST_SnapToGrid** - 2.0.0 `ST_SnapToGrid(geometry, x_resolution, y_resolution)`. GDAL 1.6.1 `GDAL 1.6.1`. `NearestNeighbor`, `Bilinear`, `Cubic`, `CubicSpline`, `Lanczos`. `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 `ST_StdDev4ma(geometry, band)`. `ST_StdDev4ma`.
- ST_Sum4ma** - 2.0.0 `ST_Sum4ma(geometry, band)`. `ST_Sum4ma`.
- ST_SummaryStats** - 2.0.0 `ST_SummaryStats(geometry, 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.
- ST_Transform** - 2.0.0 `ST_Transform(geometry, srid)`. GDAL 1.6.1 `GDAL 1.6.1`. `NearestNeighbor`, `Bilinear`, `Cubic`, `CubicSpline`, `Lanczos`. `NearestNeighbor`.
- ST_UnaryUnion** - 2.0.0 `ST_UnaryUnion(geometry)`. Computes the union of the components of a single geometry.
- ST_Union** - 2.0.0 `ST_Union(geometry)`. `ST_Union` 1 `ST_Union`.
- ST_ValueCount** - 2.0.0 `ST_ValueCount(geometry, values)`. `ST_ValueCount` (values) `ST_ValueCount` 1 `ST_ValueCount` NODATA `ST_ValueCount`.
- TopoElementArray_Agg** - 2.0.0 `TopoElementArray_Agg(topoelements)`. Returns a `topoelementarray` for a set of `element_id`, type arrays (`topoelements`).
- TopoGeo_AddLineString** - 2.0.0 `TopoGeo_AddLineString(topo, linestring, tolerance)`. Adds a `linestring` to an existing topology using a tolerance and possibly splitting existing edges/faces.
- TopoGeo_AddPoint** - 2.0.0 `TopoGeo_AddPoint(topo, point, tolerance)`. `TopoGeo_AddPoint` (split) `TopoGeo_AddPoint`.
- TopoGeo_AddPolygon** - 2.0.0 `TopoGeo_AddPolygon(topo, polygon, tolerance)`. Adds a `polygon` to an existing topology using a tolerance and possibly splitting existing edges/faces. Returns face identifiers.
- TopologySummary** - 2.0.0 `TopologySummary(topology_name)`. Takes a topology name and provides summary totals of types of objects in topology.
- Topology_Load_Tiger** - 2.0.0 `Topology_Load_Tiger(tiger_name)`. PostGIS `TIGER` `TIGER` `TIGER`.
- toTopoGeom** - 2.0 `toTopoGeom(geometry, srid)`. Converts a simple Geometry into a topo geometry.
- ~** - 2.0.0 `~(geometry, tolerance)`. A `geometry` B `geometry` TRUE `geometry`.
- ~=** - 2.0.0 `~= (geometry, tolerance)`. A `geometry` B `geometry` TRUE `geometry`.

Functions enhanced in PostGIS 2.0

- **&&** - **2.0.0** (polyhedral surface) **TRUE**. A 2D B 2D **TRUE**.

- AddGeometryColumn** - `geometry`: 2.0.0 `use typmod geometry`. `geometry` type.
- Box2D** - `geometry`: 2.0.0 `geometry`, `TIN` `geometry`. Returns a BOX2D representing the 2D extent of a geometry.
- Box3D** - `geometry`: 2.0.0 `geometry`, `TIN` `geometry`. Returns a BOX3D representing the 3D extent of a geometry.
- CreateTopology** - Enhanced: 2.0 added the signature accepting hasZ Creates a new topology schema and registers it in the topology.topology table.
- Geocode** - `geometry`: 2.0.0 `geometry` TIGER 2010 `geometry`, `geometry` `max_results` `geometry` `(geometry)` `NAD83` `geometry`. `restrict_region(geometry NULL)`.
- GeometryType** - `geometry`: 2.0.0 `geometry`, `TIN` `ST_Geometry`.
- Populate_Geometry_Columns** - `geometry`: 2.0.0 `geometry`. Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- ST_3DExtent** - `geometry`: 2.0.0 `geometry`, `TIN` `geometry`. Aggregate function that returns the 3D bounding box of geometries.
- ST_Affine** - `geometry`: 2.0.0 `geometry`, `TIN` `geometry`. Apply a 3D affine transformation to a geometry.
- ST_Area** - `geometry`: 2.0.0 `geometry` 2 `polyhedral surface` `geometry`.
- ST_AsBinary** - `geometry`: 2.0.0 `geometry`, `TIN` `geometry`. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsBinary** - `geometry`: 2.0.0 `geometry`. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsBinary** - `geometry`: 2.0.0 `geometry`. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsEWKB** - `geometry`: 2.0.0 `geometry`, `TIN` `geometry`. Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- ST_AsEWKT** - `geometry`: 2.0.0 `geometry`, `geometry`, `TIN` `geometry`. WKT(Well-Known Text) `SRID` `geometry`.
- ST_AsGML** - `geometry`: 2.0.0 `geometry`. GML 3 `'4'` `geometry`. GML 3 `TIN` `geometry`. GML 2 `GML 3` `geometry`.
- ST_AsKML** - `geometry`: 2.0.0 `geometry`. GML 2 `GML 3` `geometry`.
- ST_Azimuth** - `geometry`: 2.0.0 `geometry` 2 `polyhedral surface` `geometry`.
- ST_Dimension** - `geometry`: 2.0.0 `geometry` (polyhedral surface) `TIN` `geometry`. ST Geometry `geometry`.

- **ST_Dump** - 函数: 2.0.0 函数参考, 返回 TIN 的几何分量。Returns a set of geometry_dump rows for the components of a geometry.
- **ST_DumpPoints** - 函数: 2.0.0 函数参考, 返回 TIN 的几何分量。Returns a set of geometry_dump rows for the components of a geometry.
- **ST_Expand** - 函数: 2.0.0 函数参考, 返回 TIN 的几何分量。Returns a bounding box expanded from another bounding box or a geometry.
- **ST_Extent** - 函数: 2.0.0 函数参考, 返回 TIN 的几何分量。Aggregate function that returns the bounding box of geometries.
- **ST_Force2D** - 函数: 2.0.0 函数参考 (polyhedral surface) 函数参考。Returns "2 D" geometry.
- **ST_Force3D** - 函数: 2.0.0 函数参考 (polyhedral surface) 函数参考。Returns XYZ geometry. ST_Force3DZ 函数参考。
- **ST_Force3DZ** - 函数: 2.0.0 函数参考 (polyhedral surface) 函数参考。Returns XYZ geometry.
- **ST_ForceCollection** - 函数: 2.0.0 函数参考 (polyhedral surface) 函数参考。Returns a collection of geometries.
- **ST_ForceRHR** - 函数: 2.0.0 函数参考 (polyhedral surface) 函数参考。Returns orientation (orientation) 函数参考 (Right-Hand Rule) 函数参考。
- **ST_GMLToSQL** - 函数: 2.0.0 函数参考 (polyhedral surface) 函数参考。Returns GML geometry to ST_Geometry 函数参考。Returns ST_GeomFromGML 函数参考。
- **ST_GMLToSQL** - 函数: 2.0.0 函数参考 (polyhedral surface) SRID 函数参考。Returns GML geometry to ST_Geometry 函数参考。Returns ST_GeomFromGML 函数参考。
- **ST_GeomFromEWKB** - 函数: 2.0.0 函数参考 (polyhedral surface) 函数参考。Returns EWKB(Extended Well-Known Binary) 函数参考 ST_Geometry 函数参考。
- **ST_GeomFromEWKT** - 函数: 2.0.0 函数参考 (polyhedral surface) 函数参考。Returns EWKT(Extended Well-Known Text) 函数参考 ST_Geometry 函数参考。
- **ST_GeomFromGML** - 函数: 2.0.0 函数参考 (polyhedral surface) 函数参考。Returns GML geometry to PostGIS 函数参考。
- **ST_GeomFromGML** - 函数: 2.0.0 函数参考 (polyhedral surface) SRID 函数参考。Returns GML geometry to PostGIS 函数参考。
- **ST_GeometryN** - 函数: 2.0.0 函数参考, 返回 TIN 的几何分量。Returns ST_Geometry 函数参考。
- **ST_GeometryType** - 函数: 2.0.0 函数参考 (polyhedral surface) 函数参考。Returns ST_Geometry 函数参考。
- **ST_IsClosed** - 函数: 2.0.0 函数参考 (polyhedral surface) 函数参考。Returns LINESRING 函数参考 TRUE 函数参考。Returns (function) 函数参考 TRUE 函数参考。
- **ST_MakeEnvelope** - 函数: 2.0 函数参考 SRID 函数参考 (envelope) 函数参考。Returns SRID 函数参考 SRS 函数参考。
- **ST_MakeValid** - Enhanced: 2.0.1, speed improvements Attempts to make an invalid geometry valid without losing vertices.
- **ST_NPoints** - 函数: 2.0.0 函数参考 (polyhedral surface) 函数参考。Returns (function) 函数参考。

- **ST_NumGeometries** - 更新: 2.0.0 支持多几何体, 返回 TIN 的几何体数量。返回多几何体集合中几何体的数量。
- **ST_Relate** - Enhanced: 2.0.0 - added support for specifying boundary node rule. Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix
- **ST_Rotate** - 更新: 2.0.0 支持多几何体, 返回 TIN 的几何体。Rotates a geometry about an origin point.
- **ST_Rotate** - Enhanced: 2.0.0 additional parameters for specifying the origin of rotation were added. Rotates a geometry about an origin point.
- **ST_RotateX** - 更新: 2.0.0 支持多几何体, 返回 TIN 的几何体。Rotates a geometry about the X axis.
- **ST_RotateY** - 更新: 2.0.0 支持多几何体, 返回 TIN 的几何体。Rotates a geometry about the Y axis.
- **ST_RotateZ** - 更新: 2.0.0 支持多几何体, 返回 TIN 的几何体。Rotates a geometry about the Z axis.
- **ST_Scale** - 更新: 2.0.0 支持多几何体, 返回 TIN 的几何体。Scales a geometry by given factors.
- **ST_ShiftLongitude** - 更新: 2.0.0 支持多几何体 (polyhedral surface) 的 TIN 几何体。Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- **ST_Summary** - 更新: 2.0.0 支持多几何体。返回多几何体集合的摘要。
- **ST_Transform** - 更新: 2.0.0 支持多几何体 (polyhedral surface) 的 TIN 几何体。Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST_Value** - 更新: 2.0.0 支持多几何体 exclude_nodata_value 返回多几何体集合中的值。columnx, rowy 返回多几何体集合中的值, 返回多几何体集合中的值。exclude_nodata_value 返回多几何体集合中的值, nodata 返回多几何体集合中的值。exclude_nodata_value 返回多几何体集合中的值, 返回多几何体集合中的值。
- **ValidateTopology** - 更新: 2.0.0 支持多几何体。Returns a set of validate_topology_return_type objects detailing issues with topology.

Functions changed in PostGIS 2.0

- **AddGeometryColumn** - 更新: 2.0.0 geometry_columns 支持多几何体。geometry_columns 支持多几何体。PostgreSQL 支持 WGS84 POINT 几何体。ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326);
- **AddGeometryColumn** - 更新: 2.0.0 支持多几何体。use_typedmod 支持多几何体。
- **AddGeometryColumn** - 更新: 2.0.0 支持多几何体。geometry_columns 支持多几何体, typedmod 支持多几何体。geometry_columns 支持多几何体, typedmod 支持多几何体。

- **Box3D** - 更新: 2.0.0 新增 BOX3D 函数 BOX2D 函数。BOX2D 函数在 2.0.0 版本中新增 BOX3D 函数。新增 BOX3D 函数。
- **DropGeometryColumn** - 更新: 2.0.0 新增。新增 geometry_columns 表，ALTER TABLE 语句。ALTER TABLE 语句。
- **DropGeometryTable** - 更新: 2.0.0 新增。新增 geometry_columns 表，DROP TABLE 语句。DROP TABLE 语句。
- **Populate Geometry Columns** - 更新: 2.0.0 新增。新增 use_typmod 参数，确保 geometry columns 是定义带有类型修饰符或具有适当的空间约束。
- **ST_3DExtent** - 更改: 2.0.0 在以前的版本中这被称为 ST_Extent3D 聚合函数，它返回几何体的 3D 边界框。
- **ST_3DLength** - 更新: 2.0.0 新增 ST_Length3D 函数。新增 ST_3DLength 函数。
- **ST_3DMakeBox** - 更改: 2.0.0 在以前的版本中这被称为 ST_MakeBox3D 创建由两个 3D 点几何体定义的 BOX3D。
- **ST_3DPerimeter** - 更新: 2.0.0 新增 ST_Perimeter3D 函数。新增 ST_3DPerimeter 函数。
- **ST_AsBinary** - 更新: 2.0.0 新增 ST_AsBinary('POINT(1 2)') 函数，n st_asbinary(unknown) 不是唯一错误。新增 ST_AsBinary('POINT(1 2)::geometry'); 函数。新增 legacy.sql 文件。返回 OGC/ISO Well-Known Binary (WKB) 表示的几何/地理数据，不带 SRID 元数据。
- **ST_AsGML** - 更新: 2.0.0 新增 ST_AsGML (named arg) 函数。新增 GML 2 和 GML 3 函数。
- **ST_AsGeoJSON** - 更新: 2.0.0 新增 ST_AsGeoJSON (default arg) 函数 (named arg) 函数。返回几何或特征在 GeoJSON 格式。
- **ST_AsSVG** - 更新: 2.0.0 新增 ST_AsSVG (default arg) 函数 (named arg) 函数。返回 SVG 路径数据用于几何。
- **ST_EndPoint** - 更新: 2.0.0 新增 ST_EndPoint 函数。PostGIS 2.0.0 新增 ST_EndPoint 函数。2.0.0 新增 ST_EndPoint 函数。NULL 值。新增 ST_EndPoint 函数。ST_LineString 和 ST_CircularString 函数。
- **ST_GDALDrivers** - 更新: 2.0.6, 2.1.3 新增 - GUC 参数 gdal_enabled_drivers 参数。返回由 PostGIS 通过 GDAL 支持的栅格格式列表。只有那些 can_write=True 的格式可以由 ST_AsGDALRaster 使用。
- **ST_GeomFromText** - 更新: PostGIS 2.0.0 新增 ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') 函数。PostGIS 2.0.0 新增，SQL/MM 新增 ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') 函数。WKT 函数 ST_GeomFromText 函数。
- **ST_GeometryN** - 更新: 2.0.0 新增 ST_GeometryN 函数 NULL 值。2.0.0 新增 ST_GeometryN(...,1) 函数。ST_Geometry 函数。
- **ST_IsEmpty** - 更新: PostGIS 2.0.0 新增 ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') 函数。PostGIS 2.0.0 新增，SQL/MM 新增 ST_IsEmpty 函数。测试几何是否为空。

- [illegible]

13.12.14 PostGIS Functions new or enhanced in 1.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.5

- **&** - 1.5.0 `ST_Intersection(A, B)`. A 2D `GEOMETRY` B 2D `GEOMETRY` TRUE `BOOLEAN`.
- **PostGIS_LibXML_Version** - 1.5 `PostGIS_LibXML_Version()`. Returns the version number of the libxml2 library.
- **ST_AddMeasure** - 1.5.0 `ST_AddMeasure(geometry, start_measure, end_measure)`. Interpolates measures along a linear geometry.
- **ST_AsBinary** - 1.5.0 `ST_AsBinary(geometry)`. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST_AsGML** - 1.5.0 `ST_AsGML(geometry)`. `TEXT` GML 2 `TEXT` GML 3 `TEXT`.

- **ST_AsGeoJSON** - 1.5.0 更新内容. Return a geometry or feature in GeoJSON format.
- **ST_AsText** - 1.5.0 更新内容. 返回/输出 WKT(Well-Known Text) 格式 SRID 信息.
- **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 更新内容. 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 更新内容. Given a geometry collection, returns a multi-geometry containing only elements of a specified type.
- **ST_Covers** - 1.5.0 更新内容. Tests if every point of B lies in A
- **ST_DFullyWithin** - 1.5.0 更新内容. 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 更新内容. 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 更新内容. 返回两个几何体之间的球面距离. 1.5 更新内容. 支持 3 种模式 (longest) 模式. PostGIS 1.5 更新内容.
- **ST_DistanceSpheroid** - 1.5 更新内容. 返回两个几何体之间的椭球面距离. 1.5 更新内容. 支持 3 种模式 (longest) 模式. PostGIS 1.5 更新内容.
- **ST_DumpPoints** - 1.5.0 更新内容. 返回几何体中的点.
- **ST_Envelope** - 1.5.0 更新内容, 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 更新内容. LibXML2 1.6 更新内容. GML 格式 ST_Geometry 格式. 支持 ST_GeomFromGML 格式.
- **ST_GeomFromGML** - 1.5 更新内容. LibXML2 1.6 更新内容. 支持 GML 格式 PostGIS 格式.
- **ST_GeomFromKML** - Availability: 1.5, requires libxml2 2.6+ 支持 KML 格式 PostGIS 格式.
- **ST_HausdorffDistance** - 1.5.0 更新内容. 返回两个几何体之间的 Hausdorff 距离. 支持 3 种模式 (shortest) 模式.
- **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 更新内容. 返回几何体的长度.
- **ST_LongestLine** - 1.5.0 更新内容. 返回两个几何体之间的最长距离. 支持 3 种模式 (longest) 模式.

- **ST_MakeEnvelope** - 1.5 返回指定 SRID 的矩形包络。指定 SRID 和 SRS 的包络矩形。
- **ST_MaxDistance** - 1.5.0 返回两个几何体之间的最大距离。
- **ST_ShortestLine** - 1.5.0 返回两个几何体之间的最短距离。
- **~=** - 1.5.0 返回两个几何体是否相等。A 与 B 相等返回 TRUE。

13.12.15 PostGIS Functions new or enhanced in 1.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.4

- **Populate Geometry Columns** - 1.4.0 确保几何列具有适当的空间约束。
- **ST_Collect** - 1.4.0 将一组几何体收集到一个 GeometryCollection 或 Multi* 几何体中。
- **ST_ContainsProperly** - 1.4.0 测试 B 是否完全位于 A 的内部。
- **ST_GeoHash** - 1.4.0 返回几何体的 GeoHash 值。
- **ST_IsValidReason** - Availability: 1.4 返回文本，说明几何体是否有效，或无效的原因。
- **ST_LineCrossingDirection** - Availability: 1.4 返回一个数字，表示两条 LineStrings 的交叉行为。
- **ST_LocateBetweenElevations** - 1.4.0 返回几何体中位于指定高程 (Z) 范围内的部分。
- **ST_MakeLine** - 1.4.0 从一组几何体中创建一条 LineString。
- **ST_MinimumBoundingCircle** - 1.4.0 返回包含几何体的最小圆。
- **ST_Union** - Availability: 1.4.0 - ST_Union 被增强。ST_Union(geometry array) 被引入，并且 PostgreSQL 中的聚合函数也得到了增强。计算点集的并集。

13.12.16 PostGIS Functions new or enhanced in 1.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.3

- **ST_AsGML** - 1.3.2 将几何体转换为 GML 2 或 GML 3 格式。
- **ST_AsGeoJSON** - 1.3.4 将几何体或特征转换为 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 返回几何体的简化表示，使用 Douglas-Peucker 算法。

Chapter 14

Reporting Problems

14.1 Reporting Software Bugs

Reporting bugs effectively is a fundamental way to help PostGIS development. The most effective bug report is that enabling PostGIS developers to reproduce it, so it would ideally contain a script triggering it and every information regarding the environment in which it was detected. Good enough info can be extracted running `SELECT postgis_full_version()` [for PostGIS] and `SELECT version()` [for postgresql].

If you aren't using the latest release, it's worth taking a look at its [release changelog](#) first, to find out if your bug has already been fixed.

Using the [PostGIS bug tracker](#) will ensure your reports are not discarded, and will keep you informed on its handling process. Before reporting a new bug please query the database to see if it is a known one, and if it is please add any new information you have about it.

You might want to read Simon Tatham's paper about [How to Report Bugs Effectively](#) before filing a new report.

14.2 Reporting Documentation Issues

The documentation should accurately reflect the features and behavior of the software. If it doesn't, it could be because of a software bug or because the documentation is in error or deficient.

Documentation issues can also be reported to the [PostGIS bug tracker](#).

If your revision is trivial, just describe it in a new bug tracker issue, being specific about its location in the documentation.

If your changes are more extensive, a patch is definitely preferred. This is a four step process on Unix (assuming you already have [git](#) installed):

1. Clone the PostGIS' git repository. On Unix, type:

```
git clone https://git.osgeo.org/gitea/postgis/postgis.git
```

This will be stored in the directory `postgis`

2. Make your changes to the documentation with your favorite text editor. On Unix, type (for example):

```
vim doc/postgis.xml
```

Note that the documentation is written in DocBook XML rather than HTML, so if you are not familiar with it please follow the example of the rest of the documentation.

3. Make a patch file containing the differences from the master copy of the documentation. On Unix, type:
git diff doc/postgis.xml > doc.patch
4. Attach the patch to a new issue in bug tracker.

Appendix A

Appendix

A.1 PostGIS 3.6.1

2025/11/13

If you are upgrading postgis_topology and have topogeometry columns, make sure to run after the upgrade to fix topogeometry corruption: `SELECT topology.FixCorruptTopoGeometryColumn(schema_name, table_name, feature_column) FROM topology.layer`

A.1.1 Fixes

[#5978](#), geometry_columns needs to still parse table constraints (Paul Ramsey)

[#5987](#), ST_GeometryN fails for non-collections (Paul Ramsey)

[#5991](#), CircularString distance error (Paul Ramsey)

[#5994](#), Null pointer in ST_AsGeoJsonRow (Alexander Kukushkin)

[#5998](#), ST_Distance error on CurvePolygon (Paul Ramsey)

[#5962](#), Consistent clipping of MULTI/POINT (Paul Ramsey)

[#5998](#), [tiger_geocoder] [security] CVE-2022-2625, make sure tables requires by extension are owned by extension authored: Andrey Borodin (Yandex), reported by Sergey Bobrov (Kaspersky)

[#5754](#), ST_ForcePolygonCCW reverses lines (Paul Ramsey)

[#5959](#), [#5984](#), Prevent histogram target overflow when analysing massive tables (Darafei Praliaskouski)

[#6012](#), Remove memory leak from lwcircstring_from_lwpointarray (Paul Ramsey)

[#6013](#), [tiger_geocoder] Load Tiger 2025 data (Regina Obe)

[#5983](#), [topology] topology.FixCorruptTopoGeometryColumn to fix corruption caused by 3.6.0 upgrade (Regina Obe, Francois Bonzon)

A.2 PostGIS 3.6.0

2025/09/01

This version requires PostgreSQL 12-18beta3, GEOS 3.8 or higher, and Proj 6.1+. To take advantage of all features, GEOS 3.14+ is needed. To take advantage of all SFCGAL features, SFCGAL 2.2+ is needed.

Many thanks to our translation teams, in particular:

Teramoto Ikuhiro (Japanese Team)

Daniel Nylander (Swedish Team)

Dapeng Wang, Zuo Chenwei from HighGo (Chinese Team)

Denys Kovshun (Ukrainian Team)

A.2.1 Breaking Changes

[#5799](#), make ST_TileEnvelope clips envelopes to tile plane extent (Paul Ramsey)

[#5829](#), remove constraint checking from geometry_columns view (Paul Ramsey)

[#3373](#), [GT-255](#), [topology] Support for upgrading domains (Ayo Adesugba, U.S. Census Bureau)

[GT-252](#), ST_NumGeometries/ST_GeometryN treat TIN and PolyhedralSurface as unitary geometries, use ST_NumPatches/ST_PatchN for patch access (Loïc Bartoletti)

[#3110](#), [GT-242](#), [topology] Support for bigint (Ayo Adesugba, U.S. Census Bureau)

[#5359](#), [#5897](#), [GT-260](#) [tiger_geocoder] Use @extschema:extension@ for PG >= 16 to schema qualify dependent extensions, switch to use typmod for tiger tables (Regina Obe)

A.2.2 Removed / Deprecate signatures

[#3110](#), [GT-242](#), [topology] Support for bigint (Ayo Adesugba, U.S. Census Bureau)

[#5498](#) Drop st_approxquantile(raster, double precision), wasn't usable as it triggered is not unique error when used (Regina Obe)

A.2.3 New Features

[GH-803](#), [sfcgal] ADD CG_Simplify function (Loïc Bartoletti)

[GH-805](#), [sfcgal] Add M support for SFCGAL >= 1.5.0 (Loïc Bartoletti)

[GH-801](#), [sfcgal] ADD CG_3DAlphaWrapping function (Jean Felder)

[#5894](#), [topology] TotalTopologySize (Sandro Santilli)

[#5890](#), [topology] ValidateTopologyPrecision, MakeTopologyPrecise (Sandro Santilli)

[#5861](#), [topology] Add --drop-topology switch to pgtopo_import (Sandro Santilli)

[#1247](#), [raster] ST_AsRasterAgg (Sandro Santilli)

[#5784](#), [GT-223](#) Export circ_tree_distance_tree_internal for mobilitydb use (Maxime Schoemans)

[GT-228](#) [sfcgal] Add new functions (Scale, Translate, Rotate, Buffer 3D and Straight Skeleton Partition) from SFCGAL 2 (Loïc Bartoletti)

[raster] New GUC postgis.gdal_cpl_debug, enables GDAL debugging messages and routes them into the PostgreSQL logging system. (Paul Ramsey)

[#5841](#), Change interrupt handling to remove use of pqsignal to support PG 18 (Paul Ramsey)

Add ST_CoverageClean to edge match and gap remove polygonal coverages (Paul Ramsey) from GEOS 3.14 (Martin Davis)

[#3110](#), [GT-242](#) [topology] Support for bigint (Ayo Adesugba, U.S. Census Bureau)

[raster] Add ST_ReclassExact to quickly remap values in raster (Paul Ramsey)

[#5971](#), [tiger] Option to build --without-tiger (Regina Obe)
